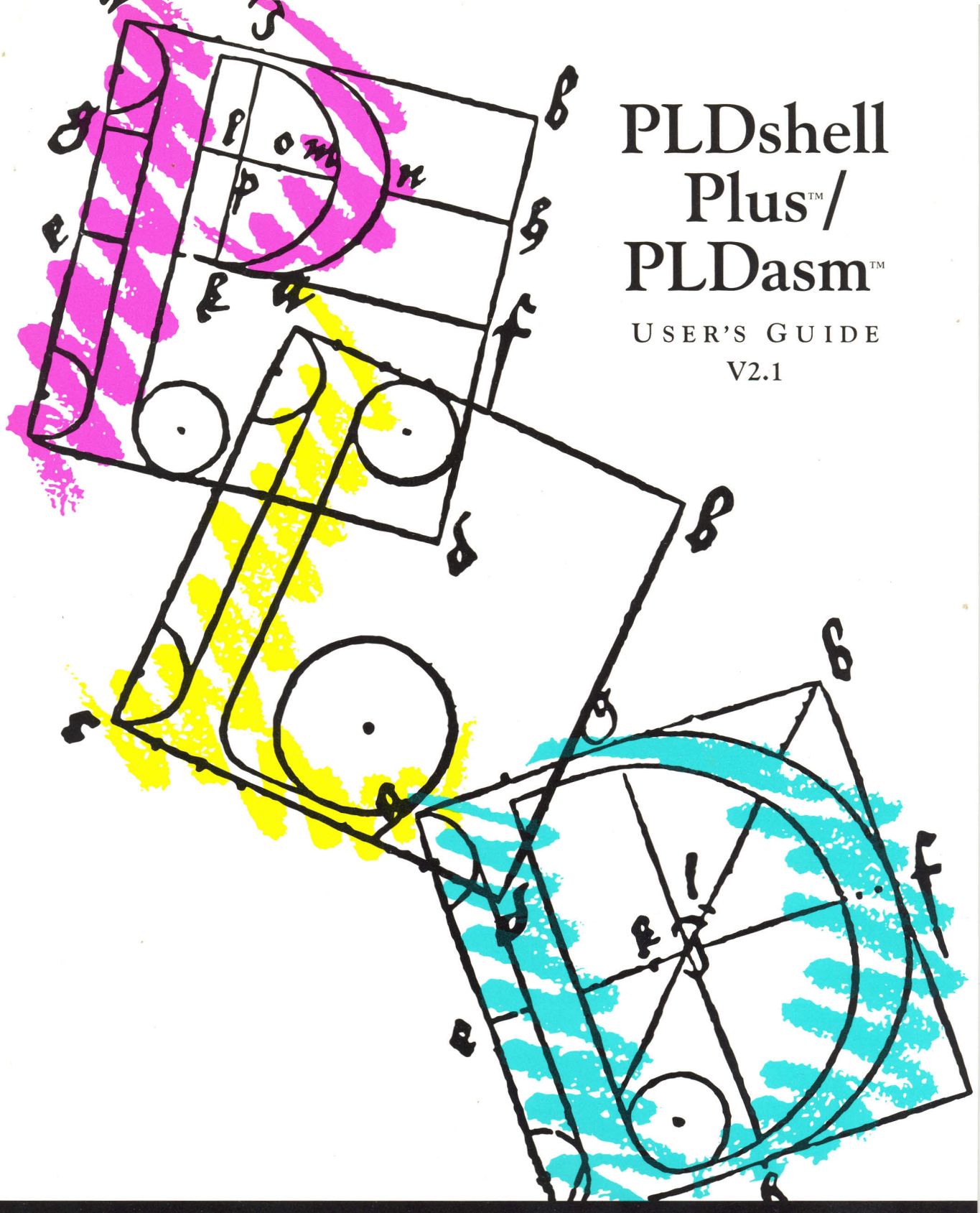


PLDshell Plus™/ PLDasm™

USER'S GUIDE

V2.1

UNIVERSAL PLD DESIGN/SUPERVISOR SOFTWARE FROM INTEL

intel

PLDshell Plus™/PLDasm™ User's Guide

V2.1

Order Number: 468816-002

Intel Corporation makes no warranty for the use of its products and assumes no responsibility for any errors which may appear in this document nor does it make a commitment to update the information contained herein.

Intel retains the right to make changes to these specifications at any time, without notice.

Contact your local sales office to obtain the latest specifications before placing your order.

The following are trademarks of Intel Corporation and may only be used to identify Intel Products:

> 376, Above, ActionMedia, BITBUS, Code Builder, DeskWare, Digital Studio, DVI, EtherExpress, ETOX, FaxBACK, Grand Challenge, i, i287, i386, i387, i486, i487, i750, i860, i960, ICE, iLBX, Inboard, Intel, Intel287, Intel386, Intel387, Intel486, Intel487, intel inside., Intellec, iPSC, iRMX, iSBC, iSBX, iWARP, LANPrint, LANSelect, LANShell, LANSight, LANSpace, LANSpool, MAPNET, Matched, MCS, Media Mail, NetPort, NetSentry, OpenNET, PRO750, ProSolver, READY-LAN, Reference Point, RMX/80, SatisFAXtion, SnapIn 386, Storage Broker, SugarCube, The Computer Inside., TokenExpress, Visual Edge, WYPIWYF.

MDS is an ordering code only and is not used as a product name or trademark. MDS is a registered trademark of Mohawk Data Sciences Corporation.

MULTIBUS is a patented Intel bus.

CHMOS and HMOS are patented processes of Intel Corp.

Intel Corporation and Intel's FASTPATH are not affiliated with Kinetics, a division of Excelan, Inc. or its FASTPATH trademark or products.

PLDshell Plus and PLDasm are also trademarks of Intel Corp.

PLDshell Plus has patents pending.

PAL and PALASM are registered trademarks of Advanced Micro Devices, Inc.

GAL is a registered trademark of Lattice Semiconductor, Inc.

The installation program used to install PLDshell Plus, *INSTALL*, is based on licensed software provided by Knowledge Dynamics Corp, Highway Contract 4 Box 185-H, Canyon Lake, Texas 78133-3508 (USA), 1-512-964-3994. *INSTALL* is Copyright © 1987-1990 by Knowledge Dynamics Corp which reserves all copyright protection worldwide. *INSTALL* is provided to you for the exclusive purpose of installing PLDshell Plus. Intel has made modifications to the software as provided by Knowledge Dynamics Corp, and thus the performance and behavior of the *INSTALL* program shipped with PLDshell Plus may not represent the performance and behavior of *INSTALL* as shipped by Knowledge Dynamics Corp. Intel is exclusively responsible for the support of PLDshell Plus, including support during the installation phase. In no event will Knowledge Dynamics Corp be able to provide any technical support for PLDshell Plus.

PLDshell Plus contains portions of Vermont Views™ software Copyright 1988, 1990 Vermont Views Creative Software. All rights reserved. Vermont Views is a trademark of Vermont Creative Software.

Table of Contents

Getting Started

Chapter 1 – Introduction

Chapter 2 – Installing PLDshell Plus/PLDasm

Chapter 3 – Using PLDshell Plus/PLDasm

Chapter 4 – PLDasm Files and Language

Chapter 5 – Compiling and Simulating

Chapter 6 – Using the Utility Programs

Chapter 7 – Device Descriptions

Chapter 8 – Design Checklist

Chapter 9 – Sample Designs

Appendix A – Language Reference Summary

Appendix B – Utilization Report

Appendix C – PLDshell Configuration File

Appendix D – Command Line Interface

Appendix E – APT Description

Appendix F – Basic PLD Information

Press <Enter> at the sign-on screen and you will see the main menu as shown in Figure GS-1. To select a menu item, use the ← and → cursor keys to highlight that item, such as **Edit**, and press <Enter>. Alternatively, you can type the first letter of the menu selection, such as 'E' for Edit.

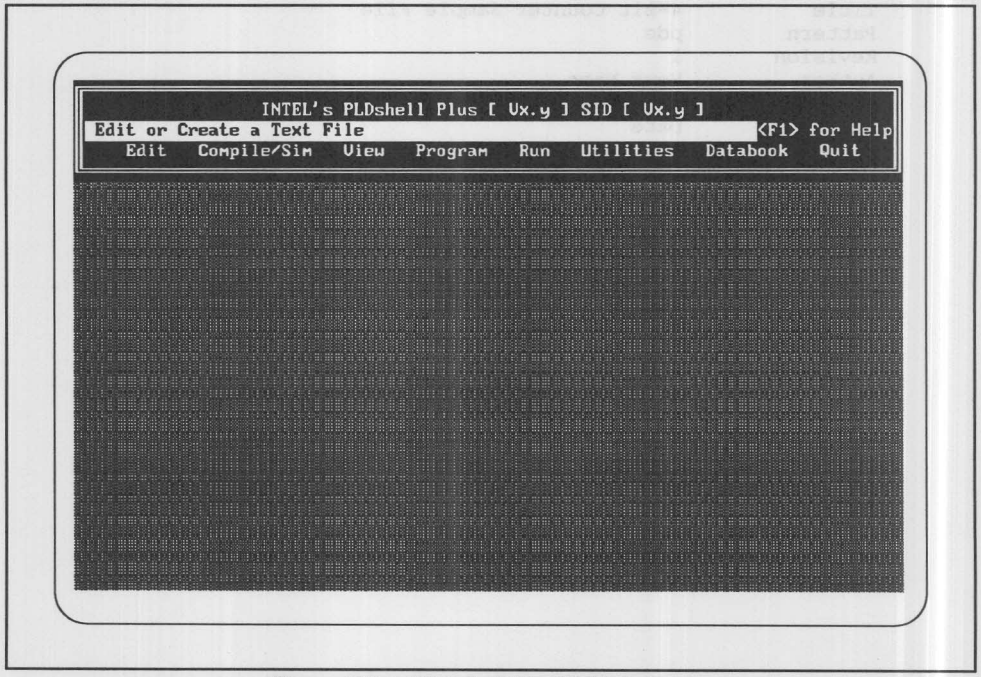


Figure GS-1. PLDshell Plus™ Main Menu

Sample Design — 4-Bit Counter

This section is a step-by-step tutorial on creating and compiling a design using PLDshell Plus/PLDasm. The example design is a simple 4-bit synchronous counter targeted for an Intel 85C224 µPLD. You can start at step 1 to create the PLDasm source file as described here and compile it to produce a JEDEC file. You can also use the sample file (4COUNT.PDS) installed with the software; in this case skip to step 6 and run the compiler. Proceed as follows:

Step 1: Header and Declarations

Open a source file using the **Edit** option. Press <F6> and enter the name of the file you wish to create with the extension .PDS then press <Enter>. Pick a base filename that does not conflict with 4COUNT.

Create header and declarations sections that contain the following information:

```
Title          4-Bit Counter Sample File
Pattern        pds
Revision       1
Author         Your Name
Company        Your Company
Date           Date

CHIP    4_count    85C224
```

Step 2: Pin Names/Assignments

Enter the pin numbers and pin names for the design as follows:

```
; pin assignments

PIN     1       CLK     ; clock pin
PIN     2       ENA     ; counter enable
PIN     15      QA      ; LSB
PIN     16      QB
PIN     17      QC
PIN     18      QD      ; MSB
```

Step 3: EQUATIONS Section

The four outputs of the counter (QA-QD) are implemented using Boolean equations. Enter the keyword "EQUATIONS" followed by the actual equations as shown below. The name on the lefthand side is the output name. The ":=" characters specify the output as a registered output. The names and symbols on the righthand side are the equations that implement the circuit. (Type the first equation as shown; the period between the Q and the A is an intentional error. It will be corrected during a later step. A sample file containing this error is shipped with the software; it is called 4ERROR.PDS)

```
; Boolean equations for registers

EQUATIONS

        QA := ENA * /Q.A

        QB := ENA * QB * /QA
            + ENA * /QB * QA

        QC := ENA * QC * /QA
            + ENA * QC * /QB
            + ENA * /QC * QB * QA

        QD := ENA * QD * /QA
            + ENA * QD * /QB
            + ENA * QD * /QC
            + ENA * /QD * QC * QB * QA
```

Step 4: Starting the SIMULATION Section

The Simulation section allows you to specify a functional simulation sequence for your designs. Enter the keyword "SIMULATION", followed by simulation commands. The first set of commands assigns a vector, specifies signals to be output to a trace file, and sets the counter inputs and registers to known states (ENA = high, CLK = low, register outputs set low). Clock and control signals should always be set before using the preload (PRLDF) command. The next set defines a loop in which the counter is clocked four times.

```
        ; set up vector and trace
        ; set to known state, preload registers (all low)

        VECTOR COUNT := [ QD  QC  QB  QA ]
        TRACE_ON  ENA  CLK  QD  QC  QB  QA
        SETF  ENA  /CLK
        PRLDF  /QA  /QB  /QC  /QD      ; clock set before
                                       ; preload command
        ; count 4 times

        FOR X := 0 TO 3 DO
               BEGIN
                      CLOCKF CLK
               END
```

Step 5: Completing the SIMULATION Section

The Simulation section is completed in two parts. First, ENA is brought low and the circuit is cycled through four clocks to test the enable signal (with ENA low the circuit resets to zero on the next clock and does not count on subsequent clocks). Then, ENA is brought high again and the counter is clocked ten more times.

```
; disable counting, then try 4 more times
SETF  /ENA
       FOR X := 0 TO 3 DO
              BEGIN
                     CLOCKF CLK
              END

; enable counting, then count 10 times

SETF  ENA
       FOR X := 0 TO 9 DO
              BEGIN
                     CLOCKF CLK
              END

TRACE_OFF

; end of simulation
```

Step 6: Running the Compiler

Save the design file, using the appropriate save command for your text editor. Press <Enter> when prompted to return to PLDshell Plus.

Select the **Compile/Sim** option from the PLDshell Plus menu (see Figure GS-2). Move to Accept and press <Enter> or press <F10> to accept the default compile and simulation conditions (the defaults are described in Chapter 5).

The PLDasm compiler runs, but since an (intentional) error exists, the compiler aborts, displaying an error message. (If you have skipped the previous steps and are using 4COUNT.PDS, this error will not be displayed. You can compile 4ERROR.PDS to generate this error if desired.)

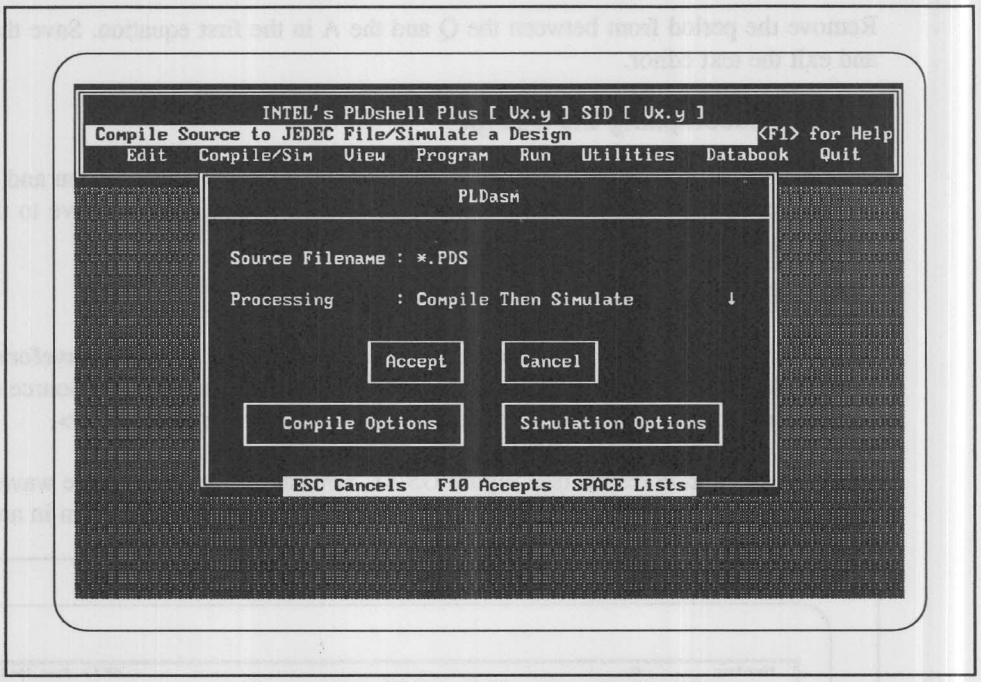


Figure GS-2. PLDshell Plus Compile/Sim Menu

Step 7: Viewing the Error File

Press ESC to move back to the main menu. Select the **View** option from the PLDshell Plus menu, then select **Error/Log Files.**

Choose the error file that has the same base name as the source file with the .ERR extension and press <Enter>. All error messages are displayed (see Figure GS-3).

```
INFO PARPDS:  Parsing file:  4ERROR.PDS.
ERROR E4304-PDSTODDB:  Invalid pin name on line 23 at ".".
ERROR E4340-PDSTODDB:  Incorrect equal operator on line 25 at "QB".
INFO PARPDS:  (0) warning(s), (2) fatal error(s).
```

Figure GS-3. Error File Listing

Move to the first error message and press <F10>. The help message for this error is displayed. The help message also recommends a course of action to correct the error.

Step 8: Revising the Source File

Press ESC three times to move back to the main menu. Use the **Edit** option to open the source file.

Remove the period from between the Q and the A in the first equation. Save the design and exit the text editor.

Step 9: Recompiling the Design

Press ESC to move back to the main menu. Select the **Compile/Sim** option and recompile the design to product a JEDEC file. Press <Enter> when done to move to the Main Menu.

Step 10: Viewing the Simulation History File

Select the **View** option from the main menu, then select **Vector/Waveform Files**. Choose the simulation history file for the design (same base name as the source file with the .HST extension) and move to Accept and press <Enter> or press <F10>.

View the simulation results (see Figure GS-4). You can move about in the waveform by using the cursor keys. The plus "+" and minus "–" keys allow you to zoom in and out.

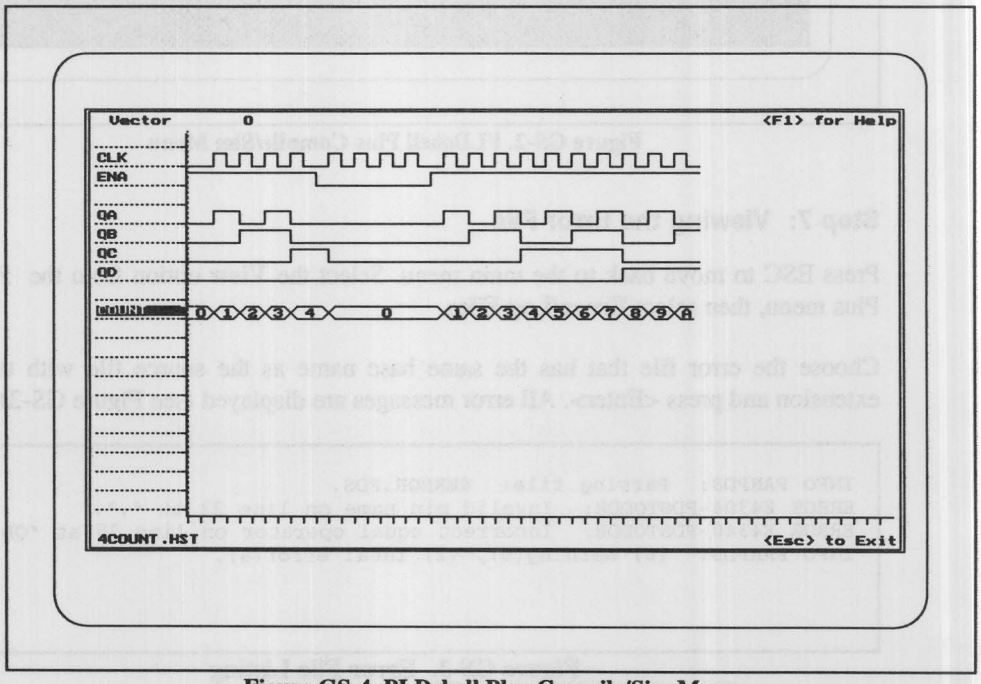


Figure GS-4. PLDshell Plus Compile/Sim Menu

Step 11: Programming a Device

Press ESC twice to return to the main menu. Select the **Program** option. If a programmer is connected to the system and programming software is installed, you can invoke the programming software from this submenu to program a device. APT (Intel's Advanced Programming Tool) is the default.

If you have not accessed this menu before, you should select your programming software using the Change Programming S/W button before attempting to program devices. Use the ↓ cursor key to move to the Change Programming S/W button and press <Enter>. When the **Configure Program** screen is displayed, with the cursor in the Programming S/W field, type the name of the programming software you want to use. If you want this to be the start-up option, move to the Save Options button and press <Enter>. Otherwise, press <F10> or move to the Accept button and press <Enter>.

Press <ESC> to return to the main menu. Press <Q> to **Quit**, press "Y" or <Enter> and return to DOS.

This ends the sample session.

Chapter 1 – Introduction

PLDshell Plus™ provides an easy-to-use menu system for PLD design that allows you to invoke Intel's PLDasm™ compiler and programming software, or your existing PLD logic compilers and programming software. Configure the menu system with the program and directory names of your existing PLD design tools and you are ready to run from PLDshell Plus. The PLDshell Plus main menu (shown in Figure 1-1) is arranged to follow the typical PLD design flow: Edit, Compile/Sim, View, and Program.

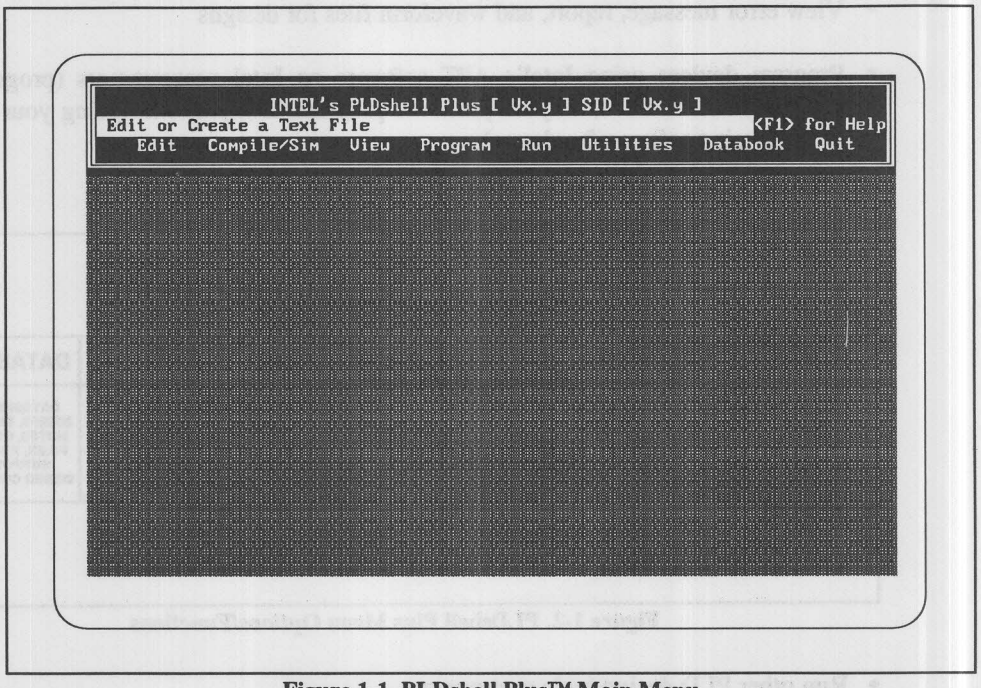


Figure 1-1. PLDshell Plus™ Main Menu

PLDasm is a logic compiler and functional simulator that runs under PLDshell Plus. PLDasm compiles PALASM®2-compatible source files to produce JEDEC files for Intel µPLDs. This allows you to use a familiar design language to evaluate the architecture of Intel µPLDs and to implement new designs.

This guide is written for experienced logic designer's who are using Intel µPLDs. If you are new to designing with PLDs, an orientation is provided in Appendix F, "Basic PLD Information."

PALASM is a registered trademark of Advanced Micro Devices, Inc.

As shown in Figure 1-2, PLDshell Plus menu options allow you to:

- Edit PLD source files using your preferred text editor.
- Compile and simulate source files to produce JEDEC programming files for Intel µPLD devices.
- View error message, report, and waveform files for designs
- Program devices using Intel's APT software on Intel programmers (programming hardware not included) or your preferred programming platform (using your existing programming software/hardware).

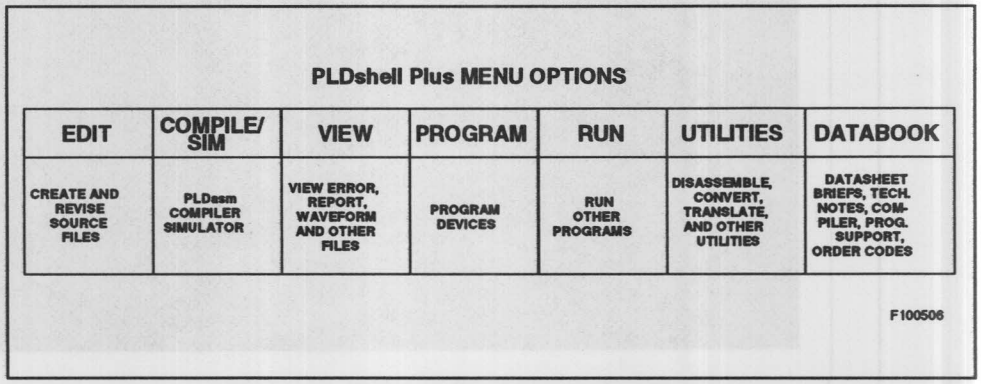


Figure 1-2. PLDshell Plus Menu Options/Functions

- Run other PLD design tools or programs.
- Run utilities that allow you to disassemble JEDEC files for supported devices into PLDasm source files for Intel µPLDs (JEDEC file to PLDasm file), or to perform a full conversion (PAL®/GAL® JEDEC file to Intel µPLD JEDEC file).
- Translate ADF/SMF files from Intel's iPLS II Programmable Logic Software into PDS files so you can take advantage of the extensive simulation benefits and ease-of-use of PLDshell Plus and PLDasm.
- View datasheet briefs for Intel µPLDs and other technical information.

PAL is a registered trademark of Advanced Micro Devices, Inc.
GAL is a registered trademark of Lattice Semiconductor, Inc.

PLDasm™ Design Compilation Overview

PLDasm compiles PALASM 2-compatible source files to produce JEDEC files for all Intel μPLDs. Figure 1-3 shows how the PLDasm compiler fits into the overall PLD design flow.

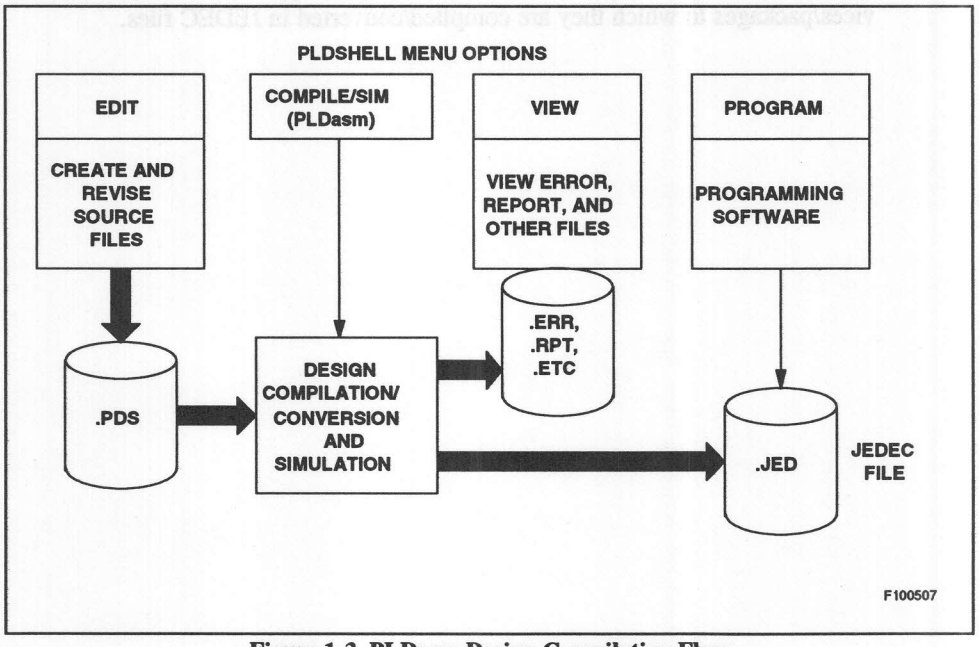


Figure 1-3. PLDasm Design Compilation Flow

The typical design process is to create the source file, compile/simulate the design to create a JEDEC file, and program devices. Error, report, and waveform files are viewed throughout the cycle. The edit/compile/simulate/view process is repeated until a design is working as desired. Devices are programmed at the end of the cycle. PLDasm offers the following features:

- Preserves your investment in learning a PLD design language and in developing source files by compiling an industry-standard language.
- Implements designs using Boolean equations, State Machine syntax, or Truth Tables (*Truth Table design is a PLDasm superset feature*).
- Functionally simulates designs.
- Maps designs into device resources and performs extensive logic minimization using the ESPRESSO mv-II* logic minimization algorithm.

* ESPRESSO mv-II is copyrighted by the University of California Regents, Berkeley

- Generates JEDEC programming files; these files include programming test vectors based on simulation output.

You can also compile .PDS files for common PAL/GAL devices using PLDasm. PAL/GAL designs are transparently converted into JEDEC files for the appropriate Intel µPLD. Table 1-1 lists the devices and packages supported in source files and the devices/packages to which they are compiled/converted in JEDEC files.

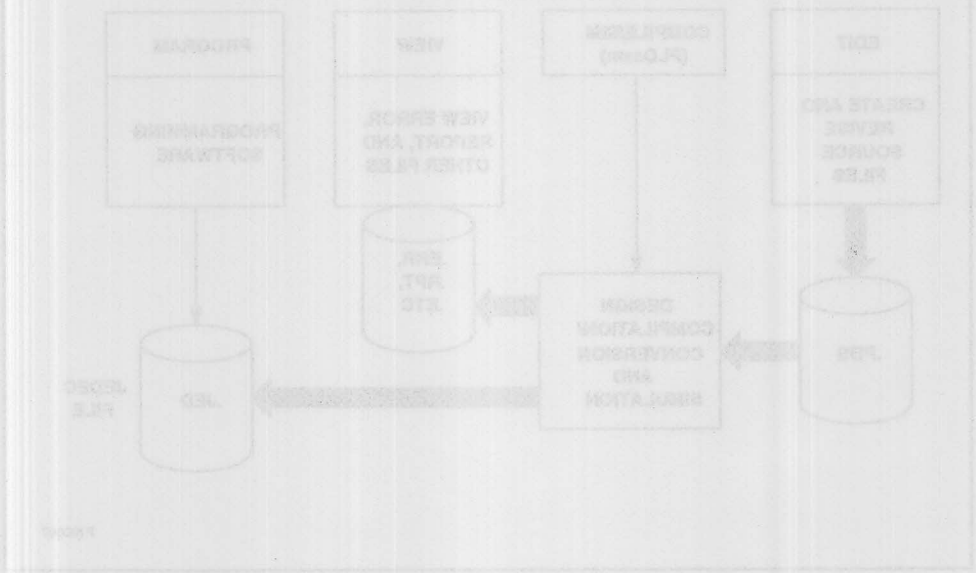


Table 1-1. Supported PLDs

Intel μPLD	Compiled To For JEDEC
iPLD610	iPLD610
iPLD910	iPLD910
iPLD22V10	iPLD22V10
85C220	85C220
85C224	85C224
85C060	85C060
85C090	85C090
85C22V10	85C22V10
85C508	85C508
5AC312	5AC312
5AC324	5AC324
5C031	5C031
5C032	5C032
5C060	5C060
5C090	5C090
5C180	5C180
Other PLDs	**Compiled To For JEDEC**
16L8	85C220
16R4	
16R6	
16R8	
16V8	
20L8	85C224
20R4	
20R6	
20R8	
20V8	
22V10	iPLD22V10
22VP10	85C22V10
iPLD16V8XP	Not supported by PLDshell Plus. Use 16V8 JEDEC file and cross programming algorithm on Data I/O or other programmer.
iPLD20V8XP	Not supported by PLDshell Plus. Use 20V8 JEDEC file and cross programming algorithm on Data I/O or other programmer.

JEDEC Disassembly Overview

Under the Utilities Menu, PLDshell Plus provides the ability to disassemble existing JEDEC files for the supported Intel µPLDs, as well as for common 20-pin and 24-pin PALs and GALs into PLDasm source files (refer to Figure 1-4). Disassembly is supported for Intel µPLDs and PAL/GAL devices listed in Table 1-1. JEDEC disassembly allows you to reconstruct source files for existing designs where the original source files have been lost, or to generate source files from existing designs to be modified for new designs.

JEDEC disassembly is available via the **Utilities — Disassemble** menu selections. Note that the source file output during the disassembly process is for the respective Intel µPLD.

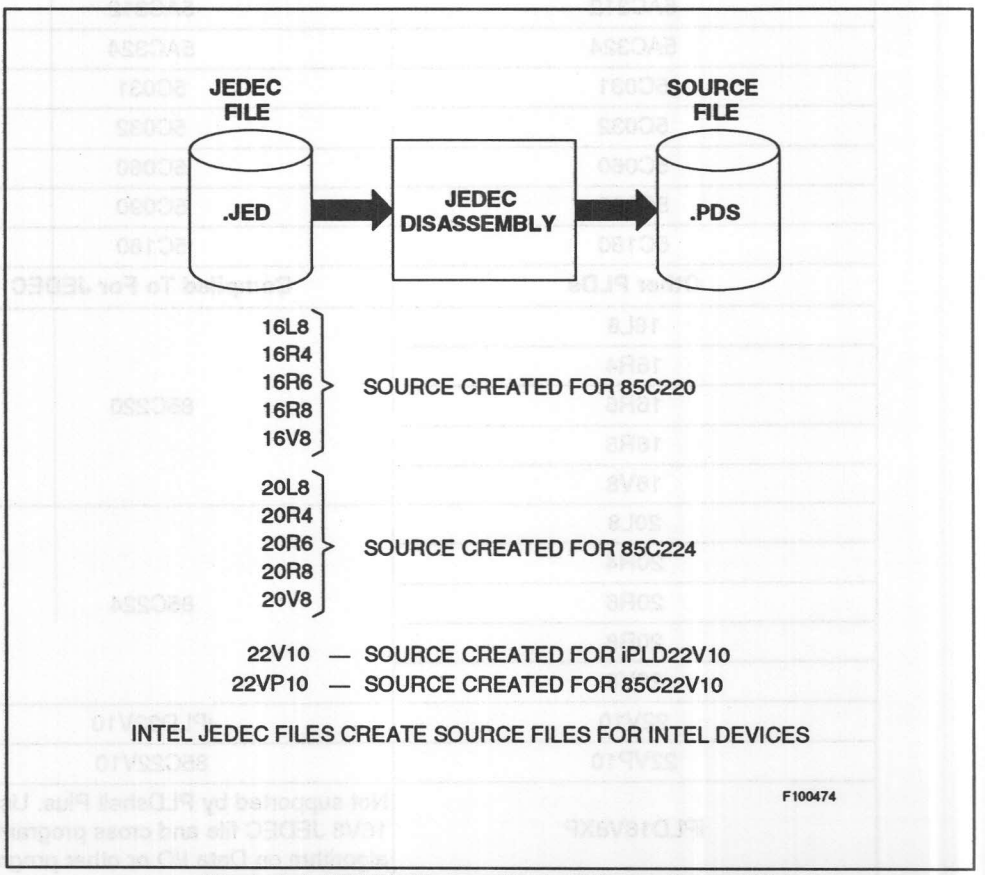


Figure 1-4. JEDEC Disassembly to PLDasm Source File

JEDEC Conversion Overview

Under the Utilities Menu, PLDshell Plus provides the ability to convert existing JEDEC files for common PALs/GALs into JEDEC files for Intel µPLDs. A PLDasm source file is automatically generated during the conversion process (see Figure 1-5). Conversion guarantees that the target Intel µPLD is functionally the same as the original design. Table 1-1 lists the devices supported by conversion.

JEDEC conversion is available via the **Utilities — Convert** menu selections.

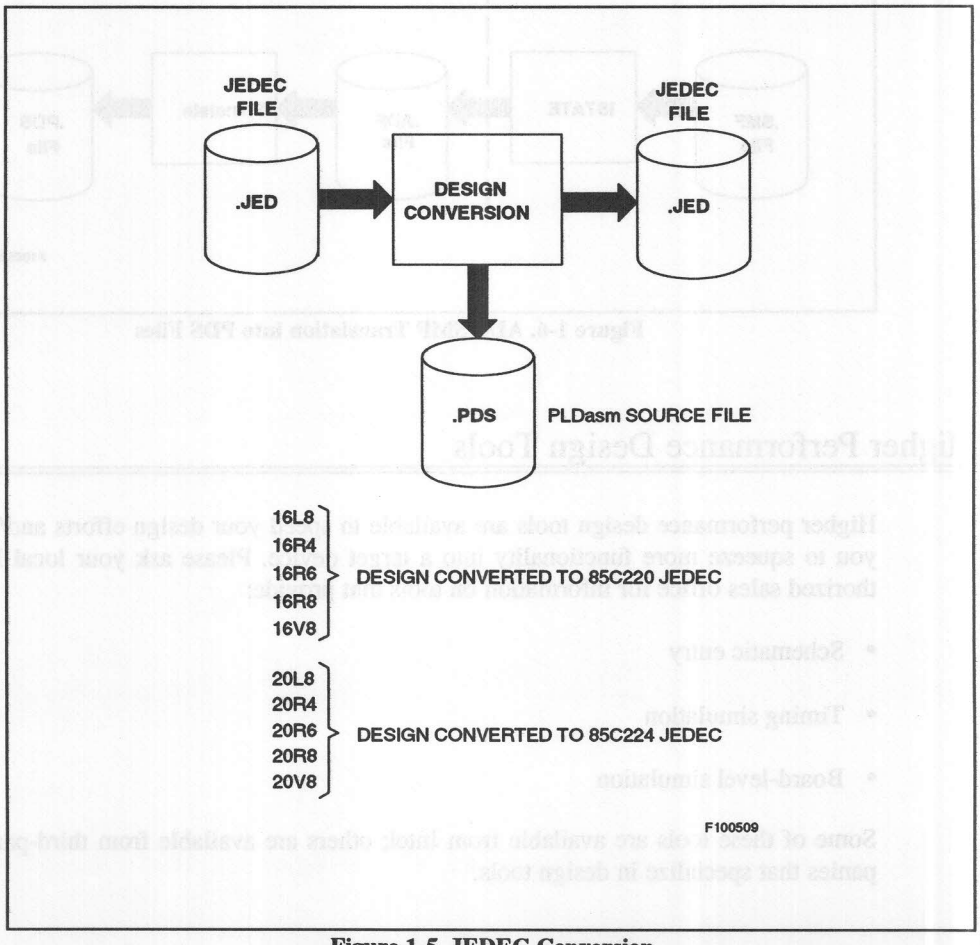


Figure 1-5. JEDEC Conversion

ADF/SMF Translation Overview

PLDshell Plus has the capability to translate ADF/SMF files into PDS files that can be compiled by PLDasm. Figure 1-6 illustrates the process. SMF files are processed through an intermediate program called iSTATE into ADF files. ADF files are then translated into PDS files. Translation is available via the **Utilities — Translate** menu selections.

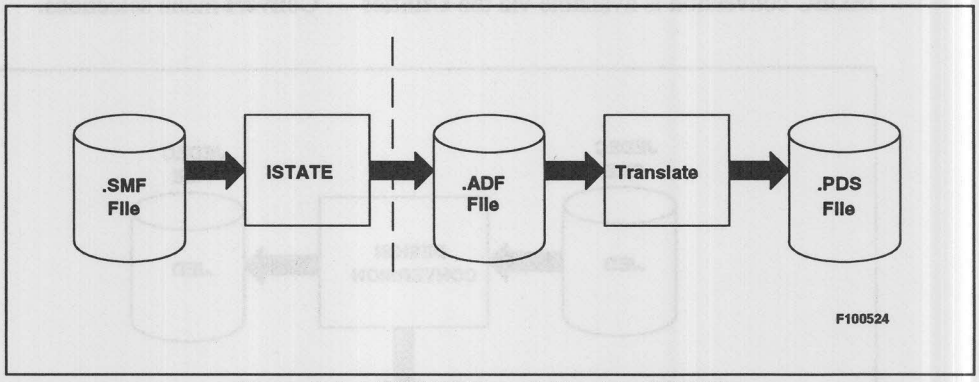


Figure 1-6. ADF/SMF Translation into PDS Files

Higher Performance Design Tools

Higher performance design tools are available to speed your design efforts and/or allow you to squeeze more functionality into a target device. Please ask your local Intel authorized sales office for information on tools that provide:

- Schematic entry
- Timing simulation
- Board-level simulation

Some of these tools are available from Intel; others are available from third-party companies that specialize in design tools.

Chapter 2 – Installing PLDshell Plus/PLDasm

System Requirements

PLDshell Plus is designed to work in systems configured as follows:

- IBM-compatible PC/AT or PS2 with 640K bytes of RAM
- MS-DOS V3.1 or later
- High-density (1.2 Megabyte) diskette drive; call Intel's EPLD Hot Line or your Intel field sales representative for other disk formats.
- Hard disk with approximately 5-Megabytes of space (4-Megabytes for installation; up to 1-Megabyte for working files while running).
- A Hercules, EGA, or VGA monitor/video card are required to view simulation vectors as waveforms.

Installation

Install PLDshell Plus as follows:

1. Insert diskette #1 into the A: drive of your system. Type:

   ```
   A:INSTALL<Enter>
   ```

2. As the INSTALL program begins executing, it displays system resources and prompts you for information such as the target drive, target directory, etc. You can press <Enter> to accept each default or can specify different names/values.
3. INSTALL prompts you for the name of your text editor. You can type the name and press <Enter> or just press <Enter> to accept the default. EDIT (DOS 5.0 editor) is the default editor name.
4. INSTALL displays messages as it expands archived files and copies those files to the target drive and directory. INSTALL prompts you to insert the next disk.
5. INSTALL prompts you before changing your AUTOEXEC.BAT and CONFIG.SYS files. If your prefer, you can skip these steps and make these changes via your text editor. The following changes should be made to the CONFIG.SYS file:

   ```
   FILES=20
   BUFFERS=15
   DEVICE=C:\PLDSHELL\STDWOUT.SYS
   ```

The number for FILES and BUFFERS can be set higher, but should not be set lower. This example assumes that C:\PLDSHELL\ is the install drive and directory. Include your actual install drive and directory.

6. The following line must appear in your AUTOEXEC.BAT file:

```
SET PLDSHELL=C:\PLDSHELL\PLDSHELL.CFG
```

   This example assumes that C:\PLDSHELL\ is the install drive and directory. Include your actual install drive and directory. Do *not* include spaces before or after the equal sign.

7. Reboot your system to load the modified AUTOEXEC.BAT and CONFIG.SYS file. You can now run PLDshell Plus/PLDasm.

Please refer to the following section if you encounter any problems installing PLDshell Plus.

Installation Notes

The following notes provide additional installation information:

1. Do *not* install PLDshell Plus in the same directory as iPLS II (Intel Programmable Logic Software II). Some executable names are the same; iPLS II executables can be overwritten.

2. If during system reboot, you encounter "Out of Environment Space" messages, you should increase your environment space. Include the SHELL command in your CONFIG.SYS file, as follows:

```
SHELL=C:\COMMAND.COM /P /E:2048
```

   where 2048 designates memory to be reserved for environment space. The number 2048 is only an example; a recommended number is 1024 bytes higher than the current number. Do not include spaces before or after the equal sign. Reboot your system to load the modified configuration file.

3. DOS recognizes only the first 127 bytes of a PATH command in an AUTOEXEC.BAT file. Please note this DOS limitation. If you are encountering search path problems when running PLDshell Plus, you should shorten your PATH command by removing some paths. You can still run programs no longer on the path by including them in the Run menu. With DOS 5.0 you can include APPEND statements in your AUTOEXEC.BAT file to search additional paths for files. Refer to you DOS User's Guide for details.

4. When installing PLDshell Plus on network systems, create local configuration files for each user. Make the PLDshell Plus install directory on the network drive

read-only to avoid accidental modifications to executable and library files. A local AUTOEXEC.BAT file, for example, should contain the following line pointing to the local configuration file (C: is the local drive):

```
SET PLDSHELL=C:\NETUSER\PLDSHELL.CFG
```

Two variables in the local PLDSHELL.CFG should point to the executable and library files residing on the network drive, as shown below (F: is the network drive):

```
IPLSPATH   F:\\PLDSHELL\\
INCLUDE  F:\\PLDSHELL\\
```

5. It is recommended that PLDshell Plus be installed over PLDshell (if you have the original product). If you do install PLDshell Plus on a system where PLDshell is also present (i.e., in a different directory), make sure you create batch files to set/reset the PLDSHELL environment variable to point to the correct configuration file. You also need to change the order of both directories in the search path to ensure proper operation. This can also be included in the batch files.

6. Note that the modifications to the CONFIG.SYS and AUTOEXEC.BAT files are made as additions to the end of the existing files. If the boot procedure already turns control of the system over to an application program via the CONFIG.SYS or AUTOEXEC.BAT file, these additions may never be executed while booting your system. If this is the case, PLDshell Plus will not run. To correct this problem, edit your files to change the order of the PLDshell Plus variables.

Configuration Notes — Windows 3.0

PLDshell Plus will function with Windows 3.0 using the DOS shell. You can do this by using the Program Manager:

1. Select File/New in the Program Manager dialog box.

2. Enter a descriptive name in the Description field.

3. Enter the command line (including the path name) that will run PLDshell Plus, in this case "C:\PLDSHELL\PLDSHELL".

4. If you wish, you can select an icon from the icons provided by Windows to use to run PLDshell Plus from the Program Manager.

NOTE

The Windows 3.0 mouse movement does not function with the PLDshell Plus menus. You must use keyboard commands and keys with PLDshell Plus under Windows 3.0.

Chapter 3 – Using PLDshell Plus/PLDasm

This section describes invoking PLDshell Plus and the various menu options.

Invoking PLDshell Plus

To invoke PLDshell Plus, from the install directory type:

```
PLDSHELL <Enter>
```

PLDshell Plus Menus

The following guidelines will help you use the menus and submenus:

- <F1> provides Help information.
- Menu options are selected (1) by using the ← and → cursor keys to highlight a menu, then pressing <Enter>, or (2) by typing the first letter of a menu option. Submenus use the ↑ and ↓ cursor keys (and <Enter>) or the first letter of the submenu.
- <ESC> backs you up one menu level at any time.
- <Enter> executes submenus that do not have options. <Enter> also accepts fields that require information such as file names.
- <F6> clears fields that include text strings (part names, file names, etc.).
- <F9> prints from the View and Databook menus.
- <F10> accepts/executes submenus.
- The space bar toggles between options where only two choices are available.
- The space bar also displays a list of options where more than two are available. Using the ↑ and ↓ keys will move the cursor through the options. Pressing <Enter> selects a highlighted option.
- Text in option fields, such as file names, can be entered and changed using alphanumeric, backspace/delete, and cursor keys.
- Home/End keys move to the top or bottom of the file in the View and Databook menus.
- PgUp/PgDn allow you to quickly scroll the screen in the View and Databook menus.

- Some menus and submenus have **Accept** and **Cancel** buttons as well as other buttons. Use the ↑ and ↓ or ← and → cursor keys to highlight these buttons and press <Enter> to activate. Pressing <F10> is the same as Accept; pressing <ESC> is the same as Cancel. Use of Accept for changing options is limited to the current PLDshell Plus/PLDasm session. To set new start-up default values, when available, select the appropriate Save button (e.g., Save Compile Options), followed by Accept or <F10>.

Main Menu

PLDshell Plus supports the following functions from the Main Menu (Figure 3-1).

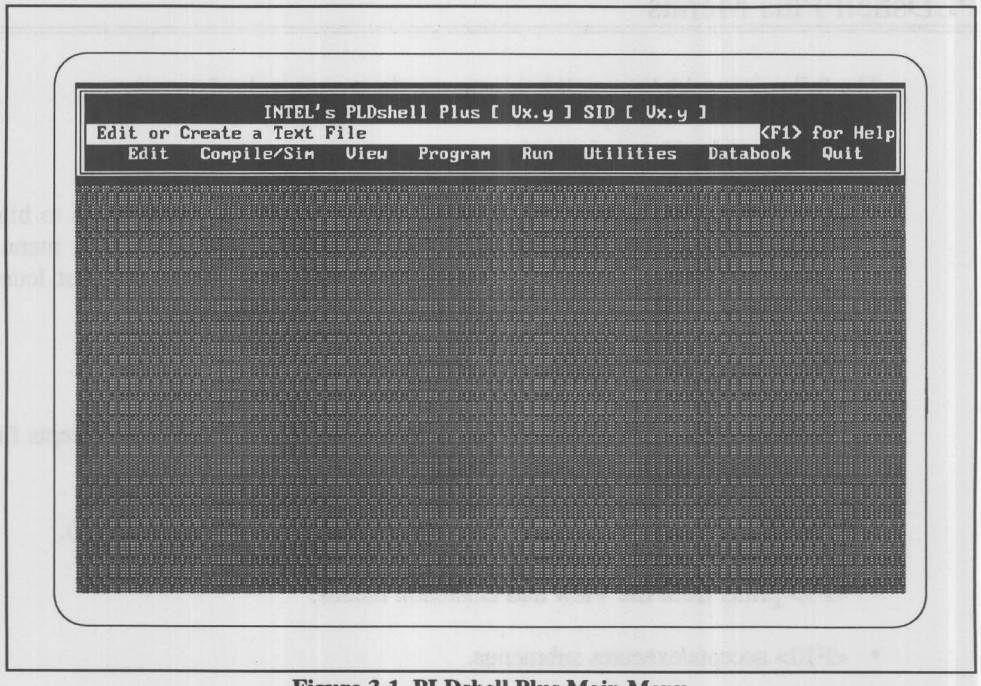


Figure 3-1. PLDshell Plus Main Menu

Edit — Edit source files or any text file using your preferred ASCII text editor.

Compile/Sim — Compile and simulate PLDasm source files to create JEDEC files for target Intel µPLDs. You can define compile/simulate options.

View — View PLD source, error, report, or simulation files to quickly locate design or fitting problems.

When viewing an error file, you can move to the desired message and press <F10> to display on-line error message help information (the error message descriptions are in the software, not in the manual).

Simulation results can be viewed in table or wave form. (Wave form display is supported on Hercules, EGA, and VGA monitors only.)

Program — Invoke Intel's APT programming software. APT commands are summarized in Appendix E, "Using APT."

Run — Run up to 24 user-defined programs including other PLD development tools. The menu is user-defined, with each menu option including default command line options and working directories.

Utilities — Provides several utility functions, including:

Disassembly of JEDEC files to PLDasm files.

Conversion of non-Intel PLD JEDEC files to Intel µPLD JEDEC files.

Translation of ADF/SMF files created for iPLS II to PLDasm files.

Listing the current directory

Changing the current directory

Invoking a DOS shell. Note that PLDshell Plus creates a swap file when creating a DOS shell. Do not delete this file or you will not be able to return to PLDshell Plus.

Modifying Options for PLDshell Plus

Databook — Allows you to view datasheet briefs on Intel µPLDs and technical notes on disassembly, conversion, etc. You may print the note being displayed by pressing <F9>.

Quit — Exit to DOS.

Edit Menu

Figure 3-2 shows the Edit menu. Use this menu to create or edit your PLDasm source files.

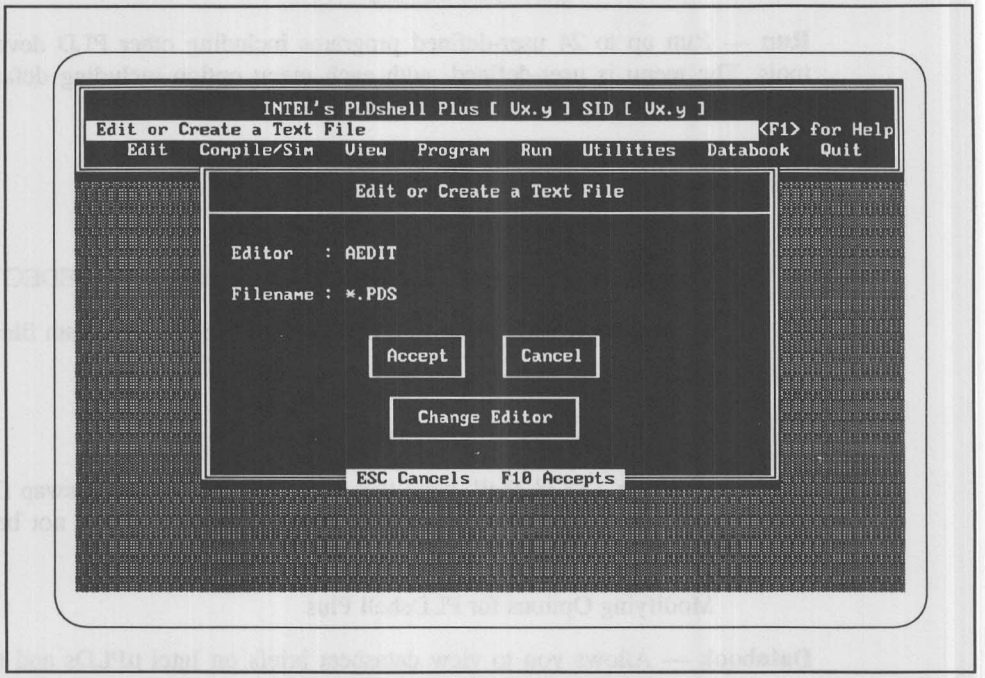


Figure 3-2. PLDshell Plus Edit Menu

Editor — Displays the currently defined text editor. A different editor can be selected using the Change Editor button. You can also temporarily set the editor by typing the editor program name in this field.

Filename — Displays the filename to be edited. The initial filename is *.PDS. The default extension of .PDS can be changed to any convenient three-character string. Pressing <Enter> when *.PDS is displayed will display a list of all files with the .PDS extension. If no files exist in the current directory, an error message will be displayed. To enter a file with no extension, type the filename followed by a period (but no extension). To invoke a text editor with no filename, press <F6> to clear the field, then accept the menu. See the Utilities Menu to change the current directory or the default filename extension.

Change Editor — Allows you to select the editor of your choice. This button displays the **Utilities—Modify Options** submenu with the cursor positioned on the Text Editor field. Type the command name of the desired editor in this field and press <F10>.

Compile/Sim Menu

Figure 3-3 shows the Compile/Sim menu. Use this menu to compile and/or simulate a logic design.

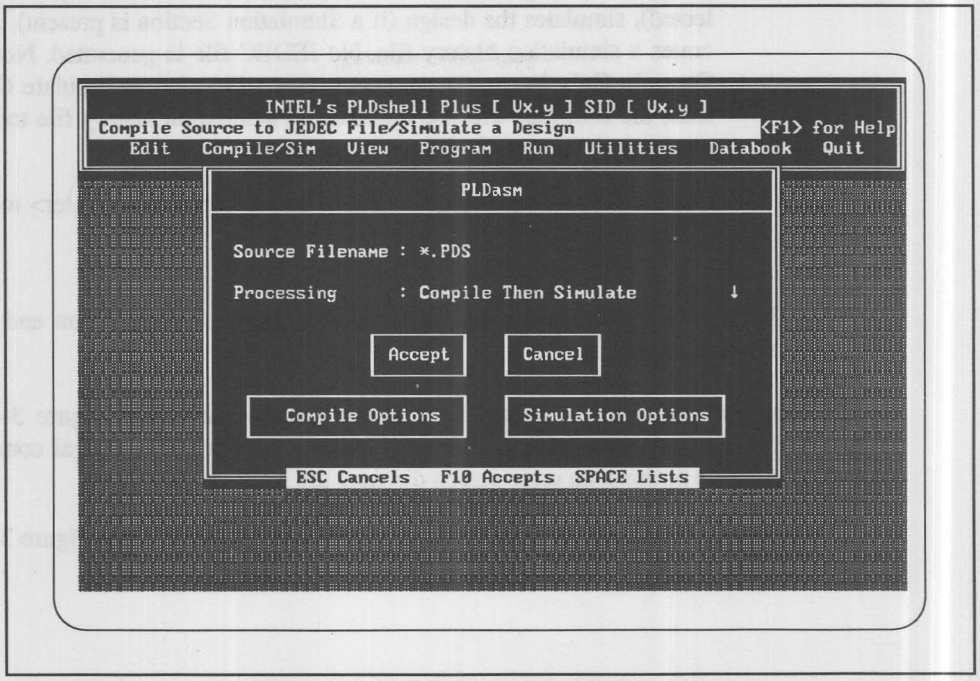


Figure 3-3. PLDshell Plus Compile/Sim Menu

Source Filename — Displays the filename of the source file to be compiled and/or simulated. The initial default is *.PDS. A source file with an extension other than .PDS can be used. If no files are in the current directory, an error message will be displayed. See the Utilities Menu to change the current directory or the default filename extension.

Processing — Press the <SPACE> key to see a list of the three processing options:

Compile Then Simulate
Compile Only
Simulate Only

Compile Then Simulate (the default) parses in the file, minimizes equations (if selected), simulates the design (if a simulation section is present), fits the design to the device resources, and generates a JEDEC programming file. Test vectors (if any) are placed in the JEDEC file based on the simulation vectors.

Compile Only parses in the source file, minimizes the equations (if selected) and generates a JEDEC file. If a Simulation Section is present, it is ignored and no test vectors are placed in the JEDEC file.

Simulate Only parses in the source file, minimizes the equations (if selected), simulates the design (if a Simulation Section is present), and generates a simulation history file. No JEDEC file is generated. Note that if Compile Only is run on the same source file after a Simulate Only session, the compiler will use the existing simulation history file to generate test vectors for the JEDEC file.

Use the ↑ and ↓ keys to highlight the desired option then press <Enter> to select.

NOTE

Refer to Chapter 5 for a task-oriented discussion of compilation and simulation.

Compile Options — Selects the Compile Options submenu (Figure 3-4). Note that some option combinations are not legal. If you select an illegal combination of compile options, an error screen displays the legal combinations.

Simulation Options — Selects the Simulation Options submenu (Figure 3-5).

Compile Options Submenu

The Compile Options submenu, Figure 3-4, allows you to set the compiler options:

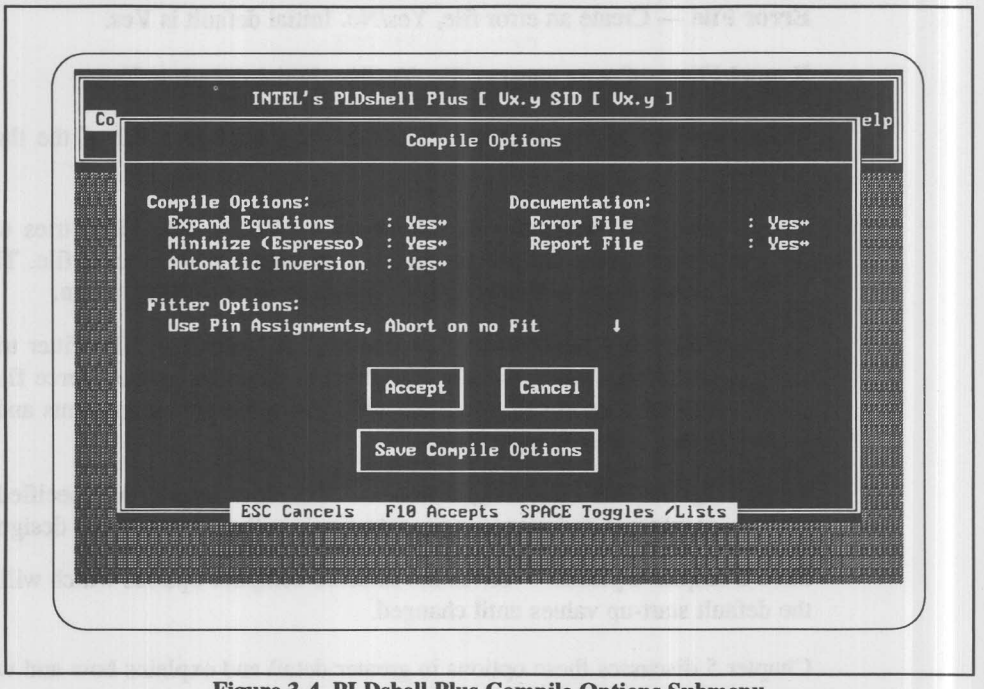


Figure 3-4. PLDshell Plus Compile Options Submenu

Expand Equations — Yes/No. Initial default is **Yes.** Expands equations to SOP (Sum-of-Products) form to allow minimization, DeMorgan's inversion, and fitting. Equations must be in SOP form for this processing. The option of not expanding is provided for designs with equations already in SOP form which the designer does not want altered in any way.

Minimize (Espresso) — Yes/No. Initial default is **Yes.** Minimizes SOP equations to least number of p-terms. PLDshell Plus uses the Espresso minimization algorithm, an algorithm that reduces equations to the least number of p-terms almost all of the time.

Automatic Inversion — Yes/No. Initial default is **Yes.** When the default is used (Yes), the parser will automatically invert equations using DeMorgan's inversion rules if the inverted form will use a smaller number of p-terms. The minimizer, when selected, will also automatically invert equations during the minimization process if the inverted form will use a smaller number of p-terms.

When the No option is selected, the parser will not invert SOP equations. The minimizer, when selected, will compute the inverted and non-inverted form and, if the inverted form is smaller, will ask the designer if it is okay to use the inverted

form. Permission to use the inverted form is requested on an equation-by-equation basis.

Error File — Create an error file, Yes/No. Initial default is **Yes.**

Report File — Create a report file, Yes/No. Initial default is **Yes.**

Fitter Options — Pressing the <SPACE> key displays a list of the three fitter options:

> **Use Pin Assignments, Abort on no Fit** — The Fitter tries to fit the design using the pin assignments specified in the source file. The Fitter aborts if any pin does not fit. This is the default fitter option.
>
> **Use Pin Assignments, But Reassign if Needed** — The Fitter tries to fit the design using the pin assignments specified in the source file. If the design does not fit, the Fitter will ignore the pin assignments and use the Fitter's auto-fit algorithms.
>
> **Ignore All Pin Assignments** — The Fitter ignores all specified pin assignments and uses its own auto-fit algorithms to attempt a design fit.

Save Compile Options — Saves the selected compiler options which will become the default start-up values until changed.

Chapter 5 discusses these options in greater detail and explains how and when you may use them.

Simulation Options Submenu

The Simulation Options submenu, Figure 3-5, allows you to set the simulation options:

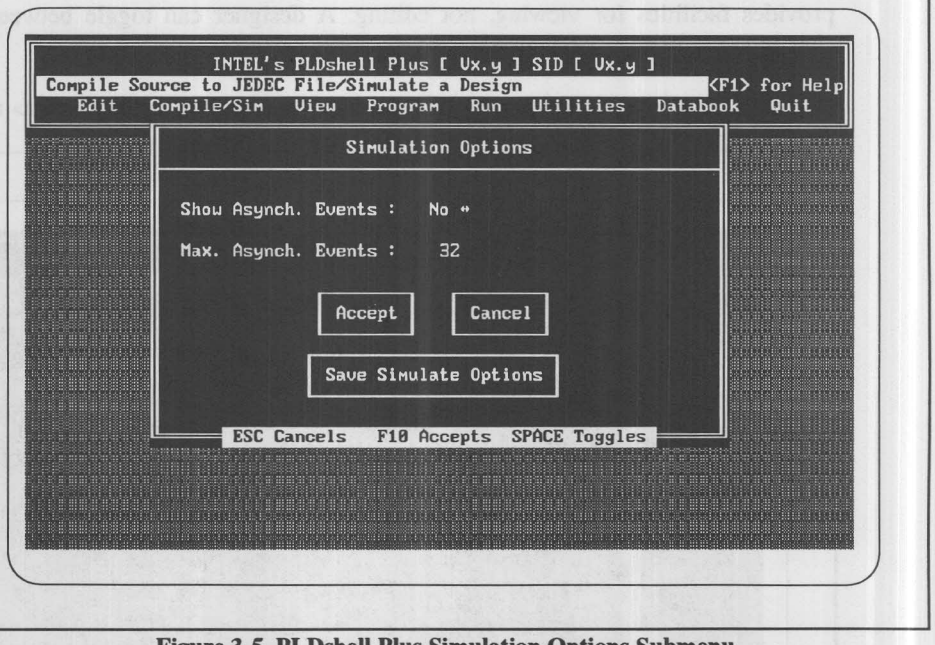


Figure 3-5. PLDshell Plus Simulation Options Submenu

Show Asynch. Events — Yes/No. Initial default is No. When set to Yes, vectors are generated for asynchronous events that occur during stabilization of each simulation cycle. This allows designers to more easily identify race conditions/glitches in combinatorial or fundamental mode sequential designs.

Max. Asynch. Events — Sets the maximum number of asynchronous events that can occur during stabilization of each simulation cycle before aborting simulation. This number can be any decimal integer in the range of 1 to 32,767. The initial default is 32.

Save Simulate Options — Saves the selected simulation options, which will become the new default start-up values until changed.

Chapter 5 describes these options in greater detail and explains how and when you may use them.

View Menu

Figure 3-6 shows the View Menu. With this menu you can view: Source Files, Error/Log Files, Report Files, Vector/Waveform Files, or Any Other File. This menu provides facilities for viewing, not editing. A designer can toggle between two open files to aid in design debugging.

Use the ↑ and ↓ cursor keys to highlight a View option, then press <Enter> to select.

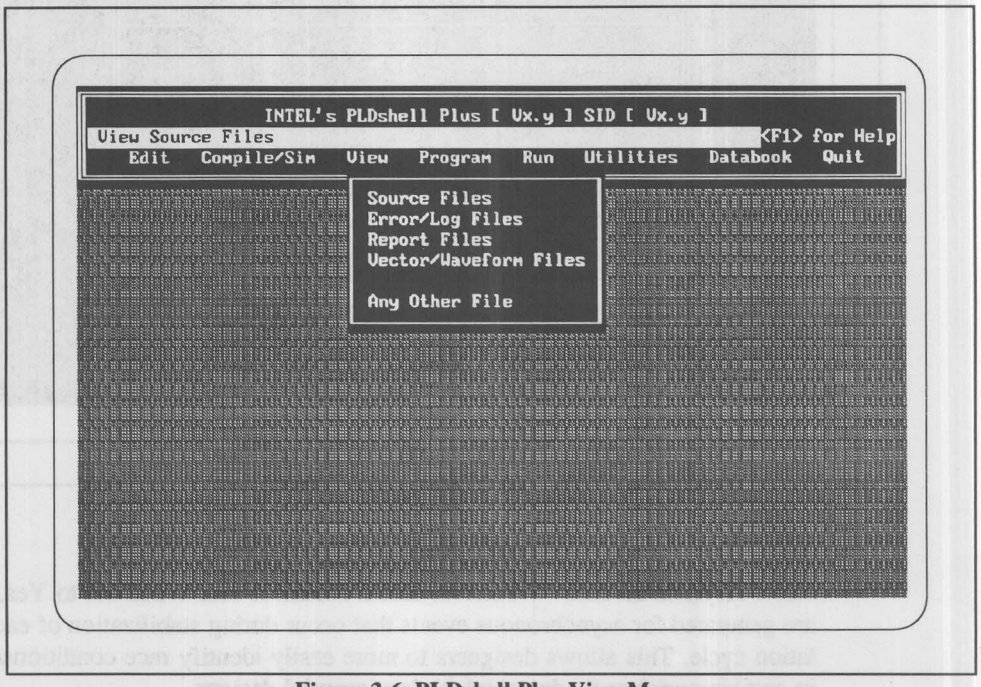


Figure 3-6. PLDshell Plus View Menu

Source Files — Displays a list of source files to view. Use the ↑ and ↓ cursor keys to highlight a file, then press <Enter> to select. The default file extension is .PDS.

Error/Log Files — Displays a list of error/log files to view. Use the ↑ and ↓ cursor keys to highlight a file, then press <Enter> to select. The file extensions are .ERR and .LOG. When viewing an error file, you can move to a specific error warning message and press <F10> to display on-line information to help you correct the error/warning.

Report Files — Displays a list of report files generated by the compiler/simulator. Use the ↑ and ↓ cursor keys to select a file, then press <Enter>. The file extension is .RPT.

Vector/Waveform Files — Displays the View Simulation Vectors submenu (see Figure 3-7 and the discussion in the next subsection).

Any Other File — Allows you to view any other file in the current directory. The initial search pattern is for all files (*.*).

Note that source, error, report, state table vectors, or other files are displayed in text format and are supported on all systems. The waveform viewer, however, can only run on systems with VGA, EGA, or Hercules graphics cards/monitors.

Toggling between two open files (in text mode) is supported by the <TAB> key. After opening the first file, you can press <TAB> to select a second file to be viewed. Use the cursor keys to select the file and press <Enter>. The <TAB> key will now toggle between the two files. To toggle between a waveform file and other files, refer to the Viewer Notes.

View Simulation Vectors Submenu

Figure 3-7 show the View Simulation Vectors submenu. This menu selects the form in which vectors will be displayed and printed.

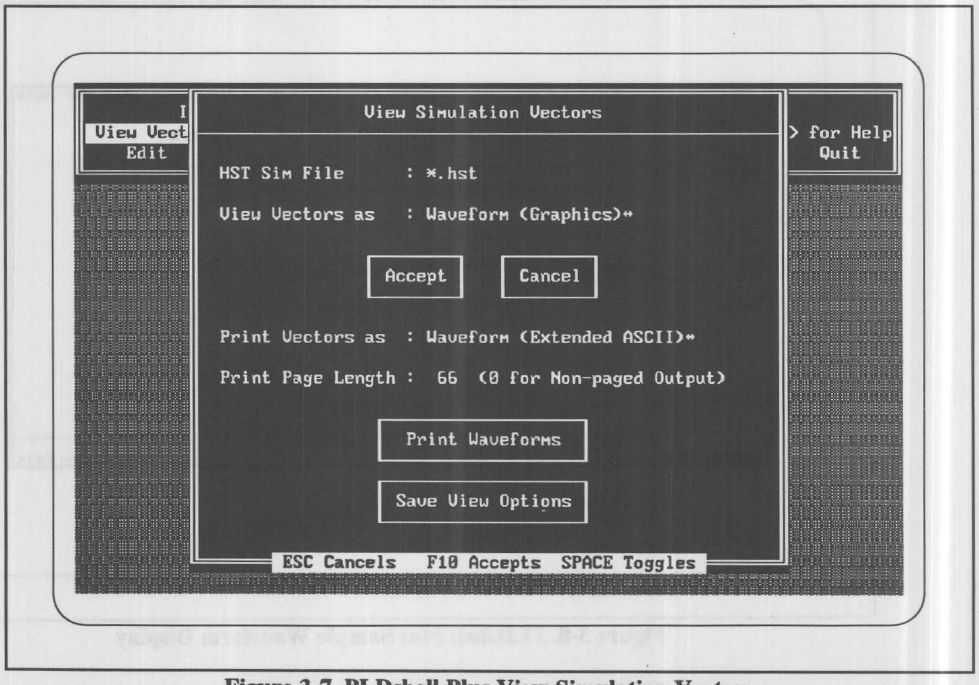


Figure 3-7. PLDshell Plus View Simulation Vectors

HST Sim File — Displays the name of a simulation file to be viewed. The initial display is *.HST if more than one file is available. Pressing <Enter>, <F10> or using the Accept button will display a list of all simulation files.

View Vectors as — Provides two viewing options: Waveform (Graphics,) or as a State Table (1s and 0s). Use the SPACE key to toggle between the two options. Figure 3-8 shows an example of the graphics viewing option.

Print Vectors as — Provides two print options: Waveform (Extended ASCII) or Waveform (Plain ASCII). Waveforms can be printed using the line-drawing character set or with standard ASCII characters.

Print Page Length — Allows you to select the page length in terms of the number of lines. The default is 66 lines per page (6 lines per inch). Enter 0 for non-paged output (continuous feed). Printer output is to the print device defined in the **Utilities – Modify Options** submenu. The default is PRN.

Print Waveforms — Prints the waveforms using the format specified in the Print Vectors As field and the Print Page Length field.

Save View Options — Saves the currently defined View Simulation Vectors options as the start-up options.

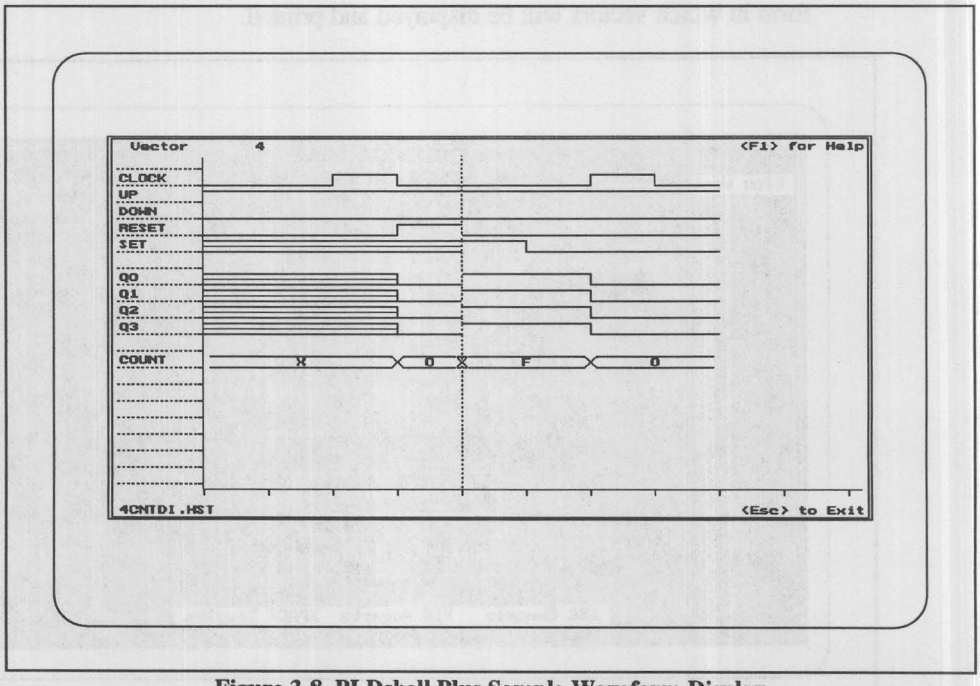


Figure 3-8. PLDshell Plus Sample Waveform Display

Graphics Waveform Viewer

PLDshell Plus provides a powerful graphics waveform viewer which allows:

- Zooming of the viewing area
- Moving, copying, or deleting a signal
- Cursor movement through either mouse or keyboard commands
- Ability to set a reference cursor to check relative timing
- Ability to save an edited waveform as a .HST file
- Ability to toggle between a waveform display and a text file.

View Waveform Commands

The View Waveform screen responds to commands from both a mouse, if a driver is installed, and a keyboard. With a two-button mouse (left and right buttons only), some commands will need to be entered with the keyboard.

Mouse Functions

Move Mouse Up	Moves cursor up one position If at top, moves signals down
Move Mouse Down	Move cursor up down position If at bottom, moves signals up
Move Mouse Left	Moves cursor left one position If at side, moves signals right
Move Mouse Right	Moves cursor right one position If at side, moves signals left
Left Button	If highlighted and held, drags signal
Left Button	If not highlighted, zooms in
Middle Button	If highlighted, copies signal
Middle Button	If not highlighted, toggles reference marker
Right Button	If highlighted, deletes signal
Right Button	If not highlighted, zooms out

Key Pad Functions

Up Arrow (↑)	Moves cursor up one position If at top, moves signals down
Down Arrow (↓)	Moves cursor down one position If at bottom, moves signals up
Left Arrow (←)	Moves cursor left one position If at side, moves signals right
Right Arrow (→)	Moves cursor right one position If at side, moves signals left
Ctrl Left Arrow	Moves cursor left half screen If at side, moves signals right
Ctrl Right Arrow	Moves cursor right half screen If at side, moves signals left
Shift Left Arrow	Moves screen left one position
Shift Right Arrow	Moves screen right one position
Home	Moves cursor to left side of screen
End	Moves cursor to right side of screen
Ctrl Home	Moves cursor to beginning of file
Ctrl End	Moves cursor to end of file
PgUp	Drag signal up one position (If highlighted) Moves cursor up ten positions (if not highlighted) If at top, moves signals down
PgDn	Drags signal down one position (if highlighted) Moves cursor down ten positions (if not highlighted) If at bottom, moves signals up
+	Zoom in
-	Zoom out
*	Toggle reference marker
r	Reset program to starting conditions
?	Display Help
INS	Copy signal if highlighted
DEL	Delete signal if highlighted
ESC or Ctrl C	Exit program
TAB	Toggles between waveform display and ASCII file
F1	Displays the On-Line Help file.

Shift F9	Save Current File. Up to the first five characters of the filename are used along with the suffix "-xx" plus the extension .HST. The number "xx" is in the range of 00 through 99. The name of the file being saved is displayed on the screen for 3 seconds.
F10	Undo up to last five edits. If no edits have been made, the system beeps.
Shift F10	Invokes the File Browser. The file browse feature allows you to display up to eight consecutive text files using the same base filename as the current waveform file but with different extensions. Pressing <Shift F10> the first time invokes the first file. Pressing <F10> after this sequences to the next file. Pressing <Shift F10> after the first occurence will back up one file. Use of the TAB key allows you to toggle between the current waveform and the current text file.

Viewer Notes

- The current cursor position is displayed at the top left of the screen. The cursor position corresponds to the vector number in the .HST file.
- The reference marker is toggled on/off by the middle mouse button or the asterisk (*).
- When a reference marker is enabled, the delta position (i.e., the cursor position minus the reference marker position) is displayed next to the cursor position. Both of these fields change to red when zoomed out past one bit of information per pixel.
- The help screen can be exited at any time by pressing the ESC key instead of paging through the whole file.
- PgUp and PgDn allows you to move through the help file.
- To toggle between the waveform display and an ASCII text file (such as the source file for the design), use the File Browse command (Shift F10) to open the source file. Then use the <TAB> key to toggle back and forth.

Program Menu

Figure 3-9 shows the Program Menu. This menu allows you to enter a programming command line to program a µPLD device. The default programmer software is APT. APT is provided and installed with PLDshell Plus/PLDasm. See Appendix E, "APT Description."

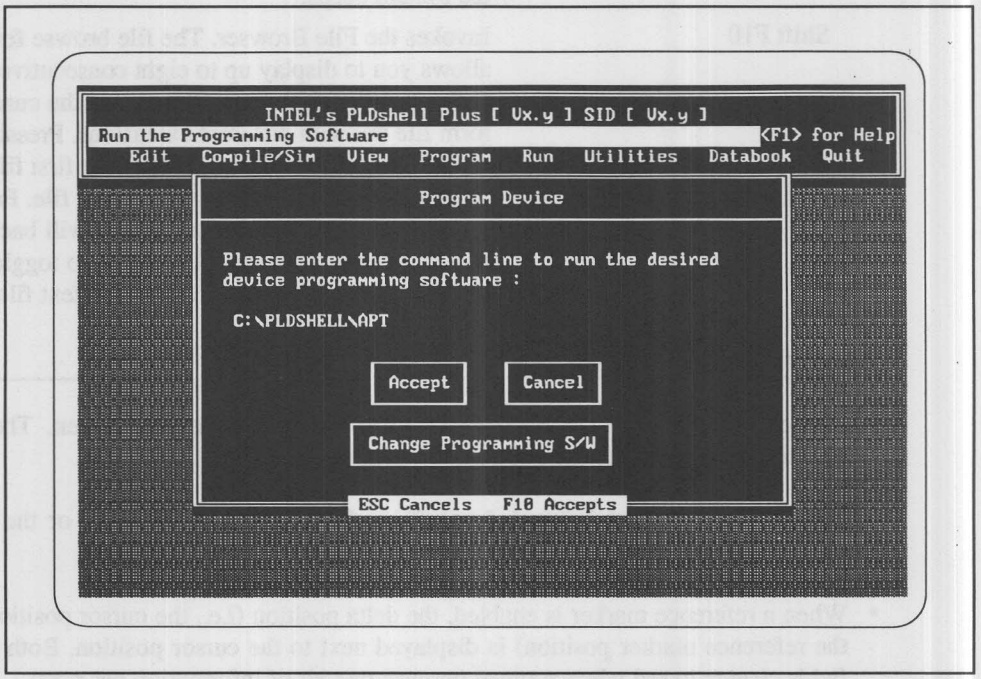


Figure 3-9. PLDshell Plus Program Menu

The initial command line is "C:\PLDSHELL\APT" with blank spaces following, allowing you to enter the appropriate commands. Three features can make your task easier:

$F uses the basename from previous processing
$D specifies the current working directory
? at the beginning of the command line results in user interaction before execution. This allows you to edit the command line if desired.

See "Run Menu Notes" for examples of how to use these features.

Change Programming S/W — Allows you to use a programming language other than APT. Using this button displays the **Utilities—Modify Options** submenu.

Figure 3-10 shows the Run Menu. This menu allows you to run up to 24 preset application programs or batch files by pressing the appropriate letter key, A through X. There are two additional letters: Y and Z. Y allows you to run an application that is not one of the other 24 available keys. Press Z to modify the program menu. Figure 3-11 shows the Modify Run submenu.

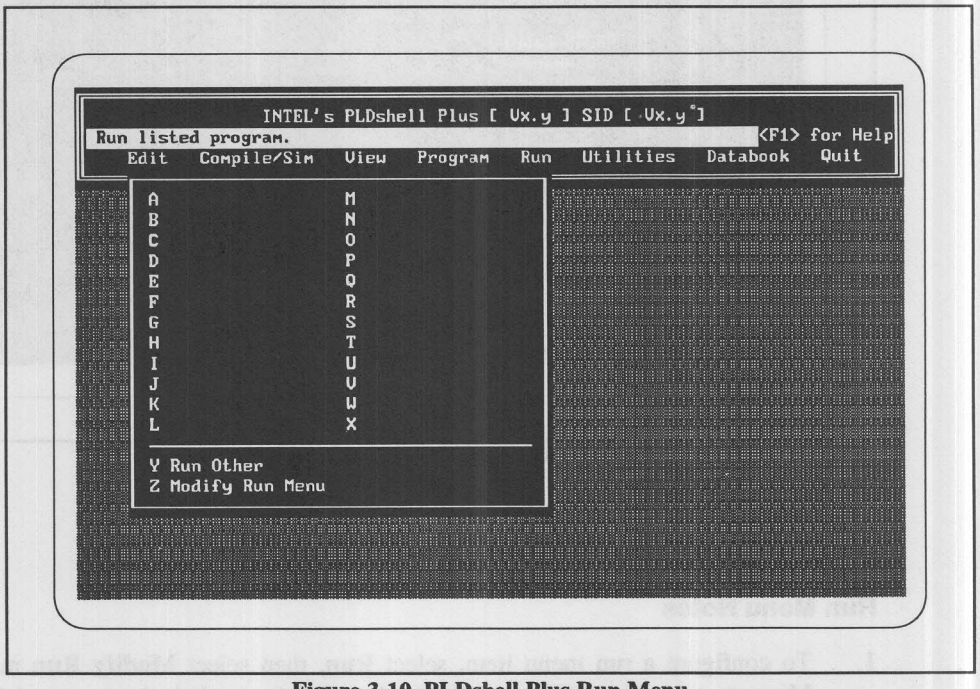


Figure 3-10. PLDshell Plus Run Menu

Modify Run Submenu

This submenu consists of three fields for each of the 24 application programs:

PROGx Label — Consists of a string of up to 16 characters that describe the application to be run.

Command — This is the command line to invoke the application program. The command line supports the following features to customize command lines:

$F uses the basename from previous processing
$D specifies the current working directory
? at the beginning of the command line results in user interaction before execution. This allows you to edit the command line if desired.

See "Run Menu Notes" for examples of how to use these features.

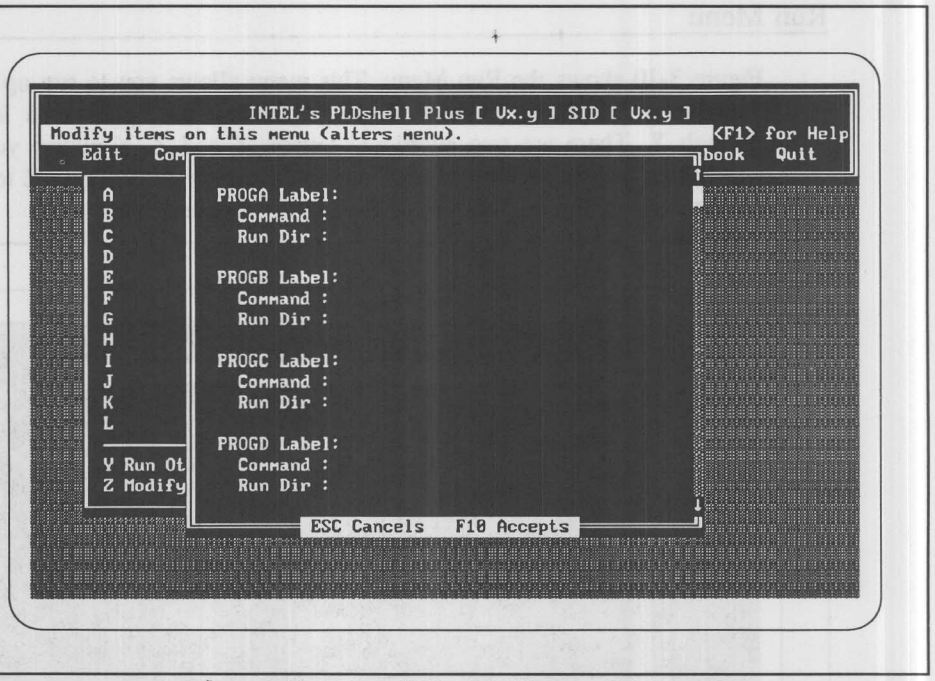


Figure 3-11. PLDshell Plus Modify Run Submenu

Run Dir — This is the DOS path for the application program.

Run Menu Notes

1. To configure a run menu item, select **Run**, then select **Modify Run** menu (Z). Move to a blank location and fill in the label (your choice), command line, and working directory fields. Press <F10> to accept the entry. You are prompted to save the changes. Answer "Y" to save the changes. Answer "N" if the changes are temporary.

2. The following is an example Run Menu entry for invoking a program that annotates JEDEC files. This illustrates Run Menu entries.

```
PROGB Label:  Annotate JEDEC
   Command:  ?AJED  $F.JED  device
   Run Dir:
```

This entry invokes a program named AJED, which requires the following command line:

```
AJED  filename.JED  device name
```

The label "Annotate JEDEC" will be displayed in the "B" position under the Run Menu since it is entered in the PROGB field. The "$F" allows the current working

base filename to be passed to the AJED program. A ".JED" extension is added by Run. The word "device" is a place holder for the device part name, which must be filled in to properly run AJED. The "?" causes Run to pause before executing the command line; this provides the opportunity to fill in the device part name.

Since no working directory is specified, the current working directory is assumed. A "$D" can be used in the command line if the working directory needs to be passed in the command line.

3. You cannot invoke Terminate-and-Stay-Resident programs (TSRs) from the **Run** menu. You will encounter "Spawn Error" messages. In some cases, the system will hang, requiring a reboot. All TSRs should be loaded *before* invoking PLDshell Plus.

Figure 3-12 shows the Utilities Menu.

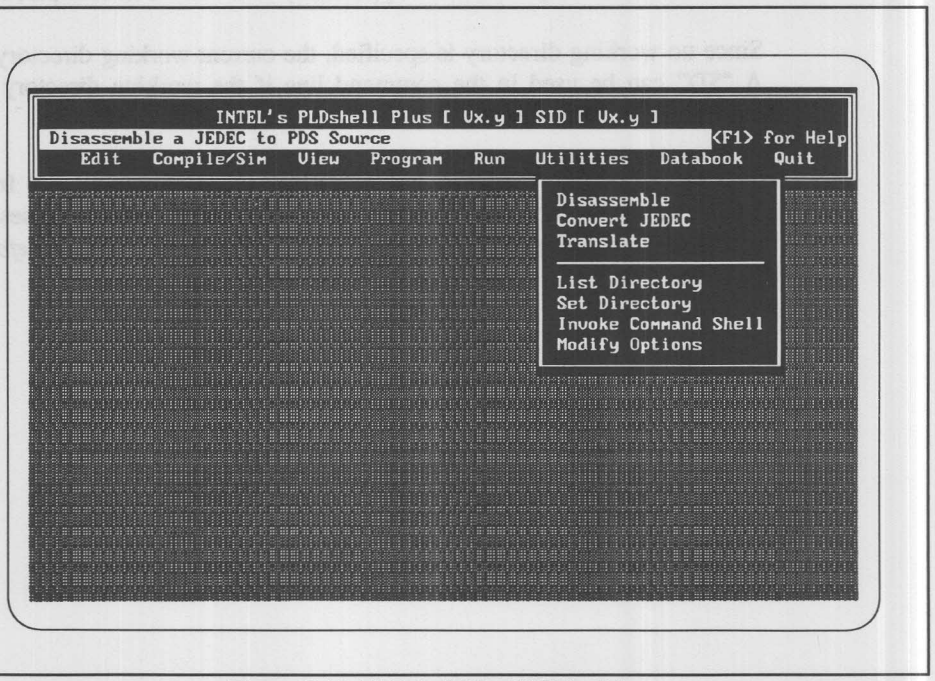


Figure 3-12. PLDshell Plus Utilities Menu

Disassemble — Disassembles existing JEDEC files (see Figure 3-13).

Convert — Converts JEDEC files for common PALs/GALs into JEDEC files for Intel µPLDs (see Figure 3-14).

Translate — Translates ADF/SMF source files to PDS source files (see Figure 3-15).

List Directory — Lists the contents of the currently defined directory.

Set Directory — Sets the directory that PLDshell Plus/PLDasm uses.

Invoke DOS Shell — Creates a DOS shell and displays the DOS prompt. Type

exit<Enter>

to return to PLDshell Plus.

Modify Options — Allows you to change default values for:

Command File
Programming SW
Text Editor
Print Device
Menu Hotkeys on or off
Source Extension

See Figure 3-16.

Disassemble Submenu

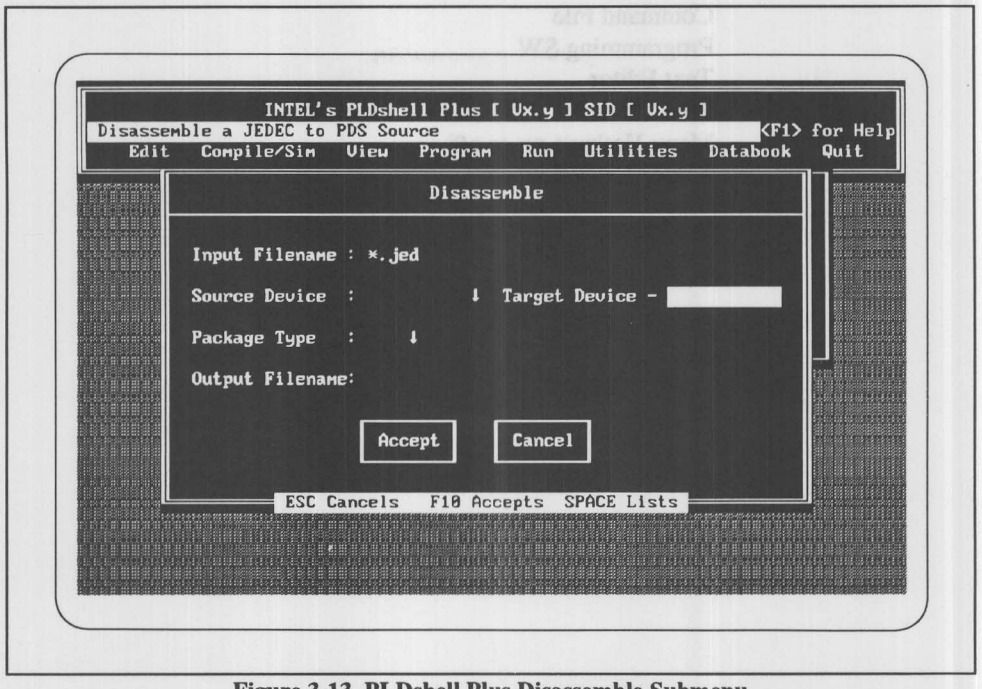


Figure 3-13. PLDshell Plus Disassemble Submenu

Input Filename — Displays a list of JEDEC files to disassemble. Use the ↑ and ↓ cursor keys to select a file name then press <Enter>. The default file extension is .JED.

Source Device — Sets the source device to be used. Pressing the <SPACE> key displays the list. Use the ↑ and ↓ cursor keys to select a device type then press <Enter>. The target Intel device will be displayed on the right of this field.

Package Type — Sets the package type. Pressing the <SPACE> key displays a list of package types for the device selected under Source Device. Use the ↑ and ↓ cursor keys to select the package type then press <Enter>.

Output Filename — Displays the output file name that will be created during disassembly. The default is the target device name plus .PDS.

Convert Submenu

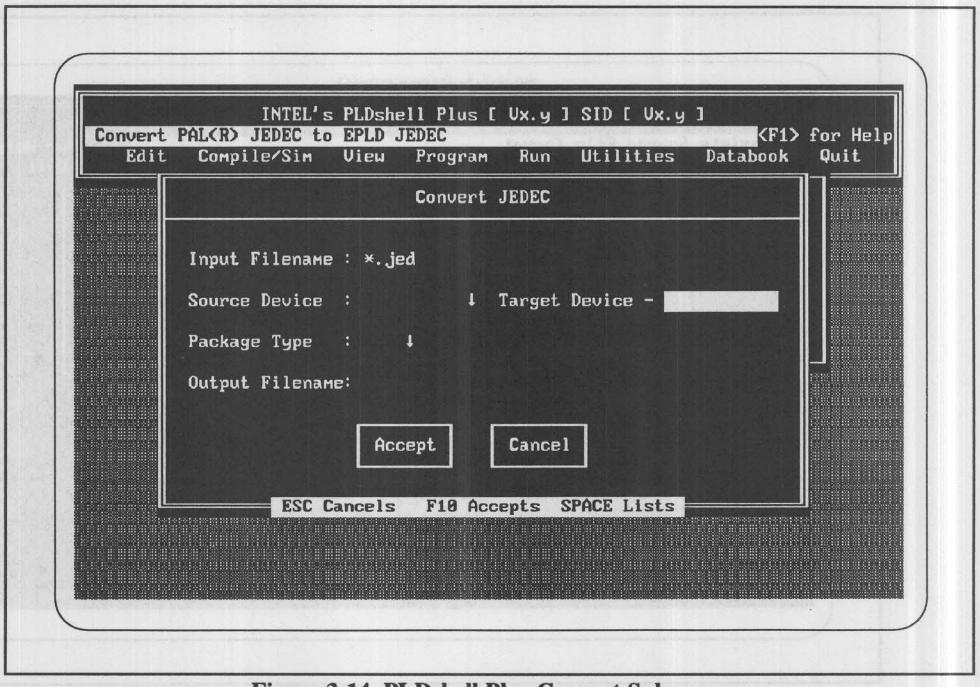


Figure 3-14. PLDshell Plus Convert Submenu

Input Filename — Displays a list of JEDEC files to convert. Use the ↑ and ↓ cursor keys to select a file name then press <Enter>. The default file extension is .JED.

Source Device — Sets the source device to be used. Pressing the <SPACE> key displays the list. Use the ↑ and ↓ cursor keys to select a device type then press <Enter>. The target Intel device will be displayed on the right of this field.

Package Type — Sets the package type. Pressing the <SPACE> key displays a list of package types for the device selected under Source Device. Use the ↑ and ↓ cursor keys to select the package type then press <Enter>.

Output Filename — Displays the output file name that will be created during conversion. The default is the target device name plus .PDS.

Translate Submenu

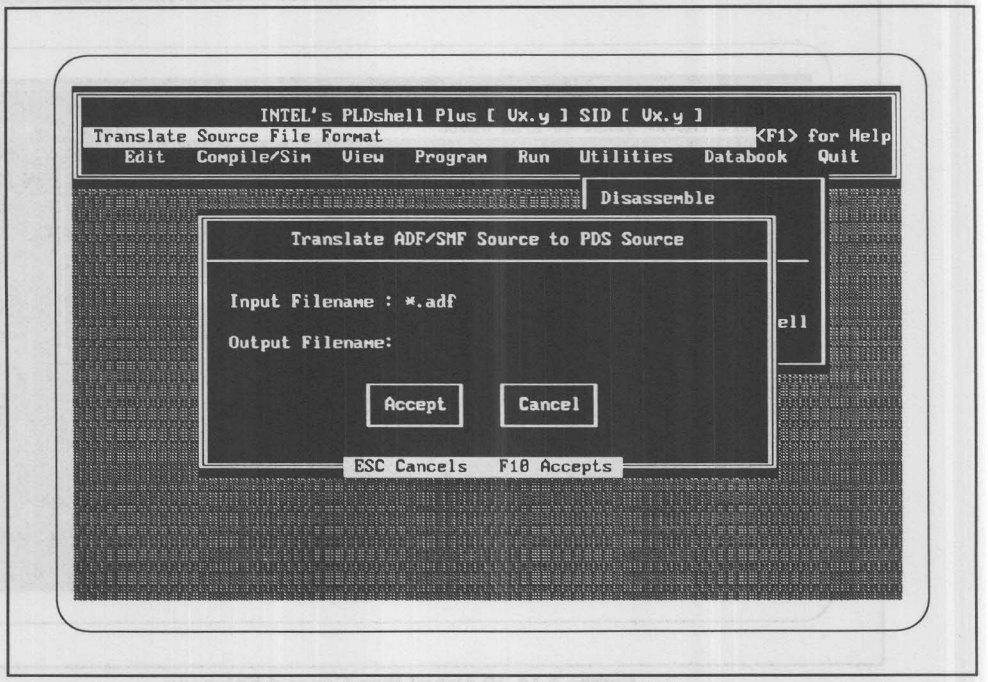


Figure 3-15. PLDshell Plus Translate Submenu

Input Filename — Displays a list of ADF files to be translated. Use the ↑ and ↓ cursor keys to select a file then press <Enter>. You can change the file type to *.SMF for iPLS II State Machine Files.

Output Filename — Displays the file name selected under Input Filename, but with a .PDS extension. If there is a file name conflict, you can press <F6> to clear this field and type in a different file name. Use an extension of .PDS.

Modify Options Submenu

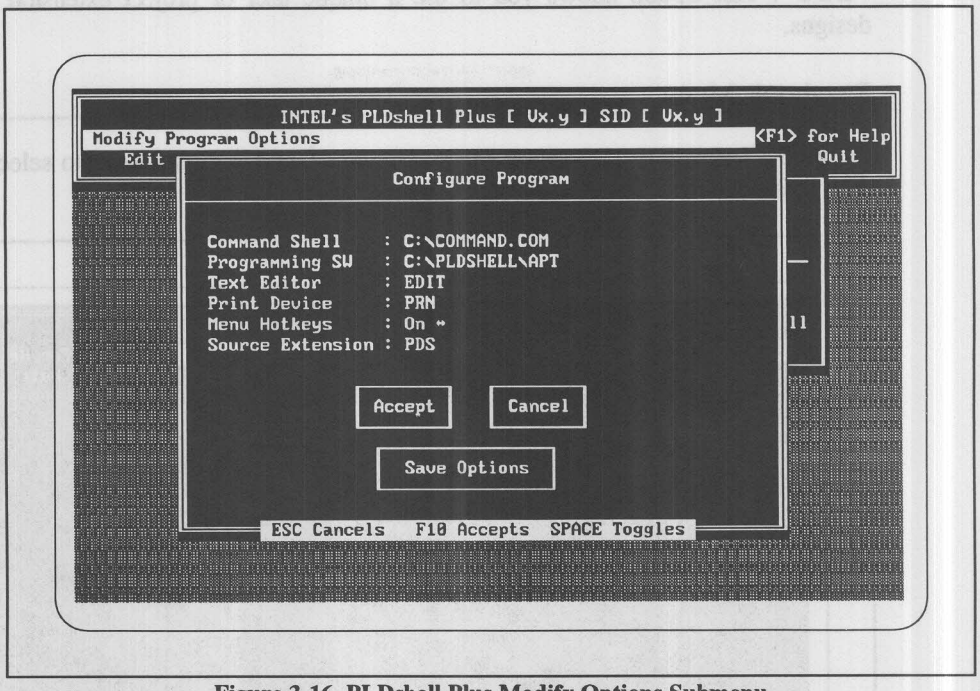


Figure 3-16. PLDshell Plus Modify Options Submenu

Command File — This is the name of the command file used by the operating system. For DOS, the default is COMMAND.COM.

Programming S/W — Allows you to select the programming software to program PLD devices. The default is APT. Can be set to whatever software runs or interfaces to your programmer. Arguments may be included in the invocation line. See "Run Menu Notes" for available arguments and how to use them.

Text Editor — Allows you to select the text editor used by the Edit Menu. The default is the editor you selected during installation (EDIT if you used the installation default). Can be set to your preferred editor. Arguments may be included in the invocation line. See "Run Menu Notes" for available arguments and how to use them.

Print Devices — Allows you to set the printer device for use from the View Menu. The default for DOS is PRN. A filename can be entered to print to a disk file.

Menu Hotkeys — Sets the menu hotkeys On or Off. Using the <SPACE> key toggles between On and Off. The default is **On**. With hotkeys On, menu selections can be made using the first letter of the menu item. When set to Off, you must use the cursor keys and <Enter> to make a selection.

Source Extension — Sets the extension for the PLDasm source file. The default is ".PDS". This option allows you to use a unique user or project extension for your designs.

Databook Menu

Figure 3-17 shows the Databook Menu. Use the ↑ and ↓ cursor keys to select a topic then press <Enter>.

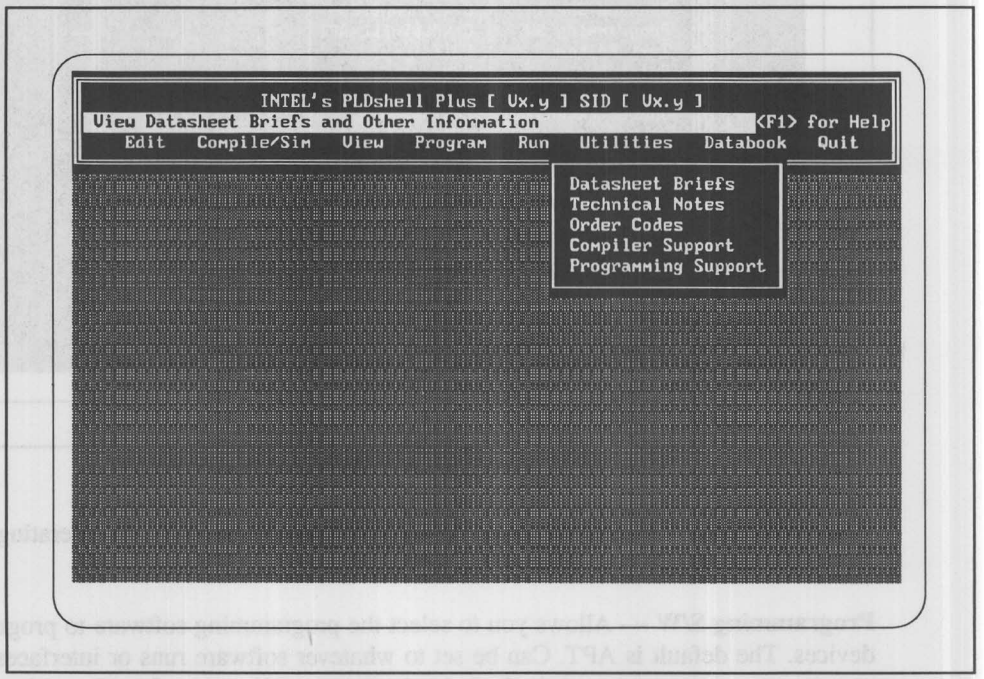


Figure 3-17. PLDshell Plus Databook Menu

Datasheet Briefs — Displays on-line information similar to that in the *Programmable Logic Databook* including: Supported Devices, Other Devices, and Software Products.

Technical Notes — Displays on-line technical information about

Device Conversion Table
Intel Part/Package Names
PDS Compilation/Conversion
JEDEC Disassembly
JEDEC Conversion
ADF/SMF Translation

Simulation
Test Vectors
Installation/Configuration

Device Order Codes — A tabular listing of Intel μPLDs and their respective order codes.

Compiler Support — A tabular listing of logic compilers/versions that support Intel μPLDs. The names, addresses, and phone numbers of each vendor are provided so you can call for the most current information.

Programming Support — A tabular listing of PLD programmers/versions that support Intel μPLDs. The names, addresses, and phone numbers of each vendor are provided so you can call for the most current information.

Chapter 4 – PLDasm Files and Language

This chapter summarizes the structure of PLDasm files and describes the syntax for implementing PLD designs.

PLDasm Files

PLDasm files are standard ASCII files containing printable ASCII characters only. The minimum PLDasm file contains the Declaration section and at least one of the following design sections: State Machine, Equation, or Truth Table section. PLDasm files may contain more than one State Machine or Truth Table sections and may contain a simulation section. Figure 4-1 shows a summary of the PLDasm file sections.

Comments

Comments can be included on any line. Comments begin with a semicolon (;). Everything between the semicolon and the end of a line is considered a comment.

The following are three examples of valid comments.

```
;PIN
   3    INB    ; THIS COMMENT FOLLOWS A PIN NAME

; THIS COMMENT IS ON A LINE BY ITSELF
```

Legal Signal Name Characters

Signal names can include 1-14 alphanumeric characters (A–Z, a–z, and 0–9) and underscore (_). PLDasm is *not* case-sensitive. *The first character must be an alphabetic character*. Any other printable ASCII character is illegal and will cause errors during compilation if they appear in signal names.

Declaration Section

The first section in all PLDasm files is the Declaration section. This section contains nine fields. The first six fields are optional and may appear in any order; these fields provide a means to identify designs and track revesions. The seventh field is optional and is used to specify the state of special bits (Turbo Bit or Security Bit) if the default state of these bits is not desired. The eighth field is required; this field selects the target device and names the pins. The last field is optional and provides a way to substitute strings to simplify designs. Figure 4-2 shows an example Declaration section. The paragraphs that follow describe the different fields. The first six fields are:

Title Field — Name of the design.

Pattern Field — Pattern number.

DECLARATION SECTION (REQUIRED) — Must be first section.

Header Information (optional)

Chip and Pin Declarations (REQUIRED)

```
CHIP        DESIGN_A      85C22V10

PIN     1       CLK
PIN     2       IN1
PIN     3       OUTA
```

String Substitutions (Optional)

```
STRING      SELA    '(INA * INB * /INC)'
```

DESIGN SECTIONS — At least one is required: State Machine, Equations, Truth Table.

STATE MACHINE SECTION (Optional)

```
S1 := BUS_REQ       ->  S2
S2 := BUS_CONT      ->  S3
    + BUS_WAIT      ->  S2
```

EQUATIONS SECTION (Optional)

```
OUTA = IN1 * IN2 * /IN3
OUTB := IN4 * IN5
      + IN6 * /IN5
```

TRUTH TABLE SECTION (Optional)

```
T_TAB   (  in1   in2     >>     /out1    out2  )
             0   0       :      0        0
             0   1       :      1        0
             1   0       :      0        0
             1   1       :      0        1
```

SIMULATION SECTION (Optional) — Must be last section

```
SETF   OE      /CLK   IN1     IN2
PRLDF      /Q2     /Q1     /Q0
SETF   IN1
CLOCKF  CLK
```

Figure 4-1. PLDasm File Section Summary

Revision Field — Version number.

Author Field — Designer's name.

Company Field — Name of your company.

Date Field — The date of the design.

Each field starts with its respective keyword and can contain a full line of text. These fields are treated as comments; you can substitute different information in these fields if desired. You may, however, use them as shown to retain compatibility with other PDS language compilers.

Options Field — The seventh field allows you to specify the state of special bits such as the Turbo Bit and Security Bit. These bits enable user-selectable device features such as low-power operation or security protection. The Options field starts with the required "OPTIONS" keyword. This is followed by the name and state of the specific options (see the Turbo and Security Bit discussions for examples). The "OPTIONS" keyword can only appear once, even when multiple options are specified as follows:

```
OPTIONS
     TURBO = ON
     SECURITY = OFF
```

Turbo Bit — Most Intel µPLDs provide a Turbo Bit that allows you to optimize a design for speed or for power savings. When the Turbo Bit is on (TURBO=ON) the device is optimized for speed. When the Turbo Bit is off (TURBO=OFF) the device is optimized for power savings and will enter standby mode in low frequency applications if no transitions are detected for a period of time. Refer to the component data sheet for additional information on the Turbo Bit.

The state of the Turbo Bit in PLDasm files is specified as follows:

```
OPTIONS TURBO = ON
```

or

```
OPTIONS TURBO = OFF
```

The "TURBO", and either "ON" or "OFF" keywords are required, as is the equal sign "=". White space may appear between any of the keywords and between the equal sign and keyword(s). If the state of the Turbo Bit is not specified, PLDasm automatically sets the Turbo Bit to On to optimize the design for speed.

Security Bit — Most Intel µPLDs provide a Security Bit (or Verify Protect Bit) that, when programmed, prevents the contents of the device from being read. This feature provides design security. The state of the Security Bit is specified like the Turbo Bit. For example,

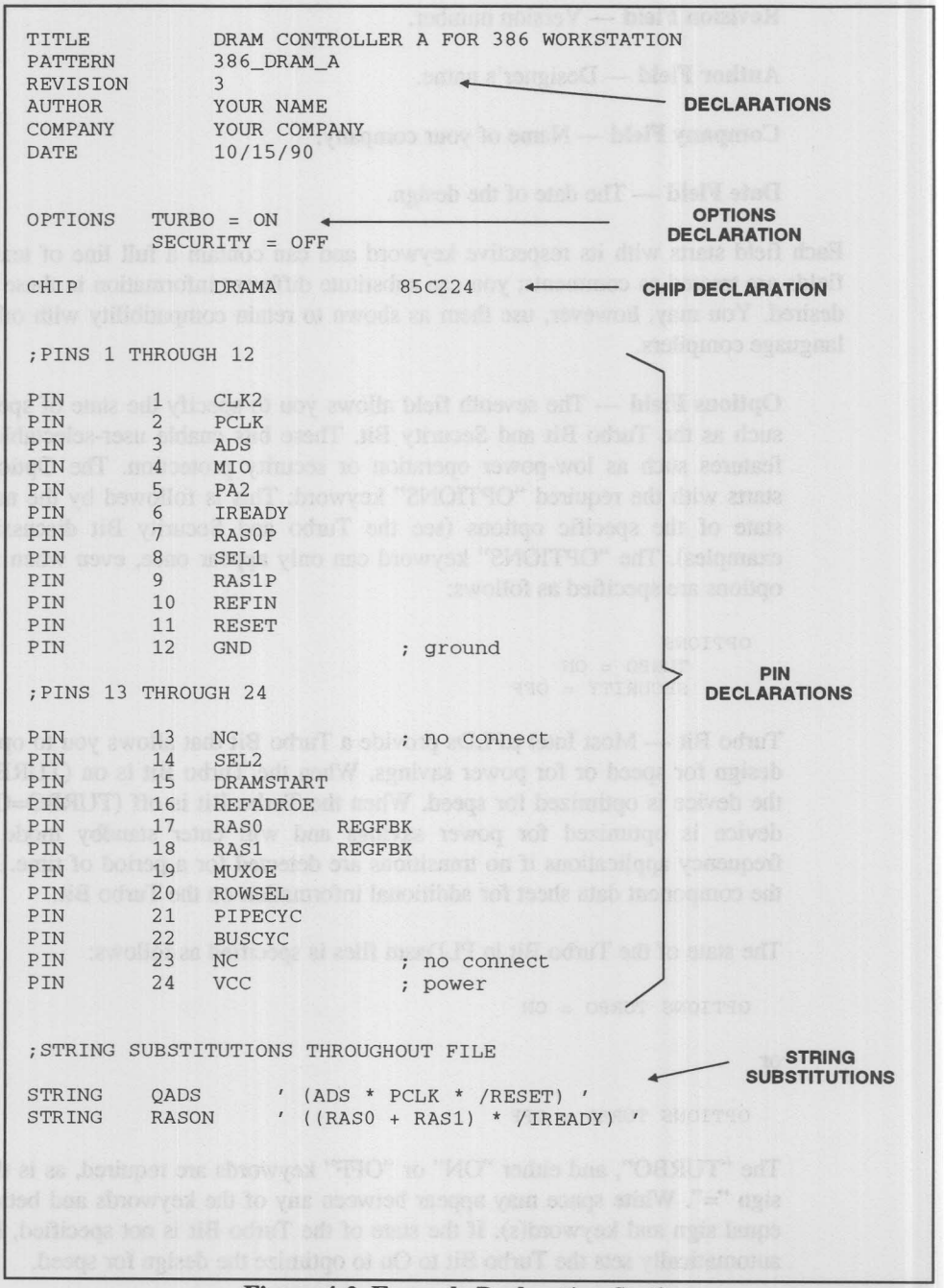


Figure 4-2. Example Declaration Section

```
OPTIONS SECURITY = ON
```

or

```
OPTIONS SECURITY = OFF
```

The "SECURITY", equal sign "=", and either "ON" or "OFF" keywords are required. If the state of the Security Bit is not specified, PLDasm automatically sets the Security Bit to OFF to allow design information to be read.

Chip Field — contains the design name, the name of the target PLD, and the names of the pins on the target device. The Chip Field begins with the keyword "CHIP". The remaining fields are defined as follows:

Design Name — The design name is required for PALASM compatibility, but is not used by PLDasm.

Device Name — The device name is required for device-specific designs and must be one of the supported PLDs. For device-independent design, the reserved word "INTEL_ARCH" can be used. Chapter 5 describes the differences between device-specific and device-independent design. Table 7-1 (Chapter 7, "Device Descriptions") lists the devices, packages, and device names. For non-Intel PLDs (PALs/GALs) , Table 7-1 also lists the Intel µPLD into which the design is fit.

Pin Names — Names the input and output signals. The names can include 1-14 alphanumeric characters; the first character must be an alphabetic character. Pin names can be separated by spaces (see the figure), commas, or carriage returns. The first example shows the easiest method of defining pins. The order of the pins is not important with this method.

```
PIN  1    ADDR0
PIN  2    ADDR1
PIN  4    ADDR3
PIN  5    ADDR4
PIN  3    ADDR2
```

When using this method of pin declarations, a keyword can be added after the signal name to define the architecture of I/O pins on devices that offer additional I/O options. This is described later in the "Using Additional Features" section.

A second method for defining pin names is shown below. If this method is used and a particular pin is not used in the design (no connect), it must be designated "NC". Pin names must appear in sequential order (first name for pin 1, second name for pin 2, etc.) with this method.

```
ADDR0, ADDR1, ADDR2, NC
ADDR3
NC
ADDR4, OUT2, NC, OUT5
```

PLDshell Plus is capable of automatically assigning inputs and outputs to pins in many cases. Refer to the "Automatic Pin Assignments" section later in this chapter.

String Field — Allows global string substitutions in PLDasm files. Each String substitution begins with the keyword "STRING" followed by the name of the string and the substitution value. In Figure 4-2, string substitutions are used to group some signals that are frequently used together. This can help simplify the Equations section. It is recommended that all text in substitution strings be enclosed in parentheses to improve readability in output files. This will ensure correct results when used with other Boolean operators.

Basic Circuit Design Using Boolean Equations

This section describes basic combinatorial and registered circuit design for Intel μPLDs using Boolean equations. Advanced registered design topics, such as asynchronous clocking of registers, use of T-type, JK-type, and SR-type flip-flops, Automatic Pin Assignments, Alternate I/O Options, Truth Tables, and State Machines are covered in later sections.

A complete list of PLDasm operators (combinatorial and registered) is as follows:

Operator	Description
/	Active-Low in pin declaration; Boolean NOT elsewhere in file
*	Boolean AND
+	Boolean OR
:+:	Boolean XOR
=	Combinatorial Output
*=	Latched Output
:=	Registered Output

Combinatorial Circuits

Combinatorial circuits are easily implemented in PLDasm files using Boolean equations. These equations must be placed in the Equations section of PLDasm files. An Equations section begins with the keyword "EQUATIONS". The output from the equation is a pin name or node name elsewhere in the design. Figure 4-3 shows some simple logic functions implemented using Boolean equations.

The first seven equations each implement basic logic gates. The eighth equation (SUM1) is a slightly more complex logic function with three AND terms (product terms, or p-terms) feeding one OR gate. The last equation (SUM2) shows how the output from

```
EQUATIONS

     AND1      = IN1 * IN2                ; LOGICAL AND

     /NAND1    = IN1 * IN2                ; LOGICAL NAND

     OR1       = IN1 + IN2                ; LOGICAL OR

     /NOR1     = IN1 + IN2                ; LOGICAL NOR

     A         = /IN                      ; LOGICAL NOT

     XOR1      = IN1 :+: IN2              ; EXCLUSIVE OR

     XOR2      = IN1 * IN2 :+: INA * INB + INC

               ; EXCLUSIVE OR - PRECEDENCE IS
               ; ((IN1 * IN2) :+: (INA * INB)) + INC

     SUM1      = IN1 * IN2 * IN3
               + /IN1 * (/IN2 * IN4)
               + IN1 * IN5 * IN6          ; SUM OF PRODUCTS

     SUM2      = IN1 * IN2 * /SUM1
               + IN1 * /IN2 * /SUM1
               + /IN1 * IN5 * /SUM1       ; SUM OF PRODUCTS
```

Figure 4-3. Combinatorial Circuits Using Boolean Equations

one equation can be fed back and used to qualify other equations. In this example, the SUM1 output is fed back through the logic array to help qualify SUM2.

Note that parentheses can be used to specify precedence in Boolean equations. SUM1 shows an example of this. Equations can be expressed in Sum-of-Products form, but this is not a requirement. PLDasm can convert equations into Sum-of-Products form automatically, unless the Expand Equations option in the Compile submenu is turned off (Expand Equations = No).

Active-High/Active-Low Outputs

As described in the device section of this guide, macrocells on most Intel devices contain an inversion control bit that allows the respective output to be configured as active high or active low. An output is configured as active low when *either* the pin name *or* equation feeding the pin includes the slash prefix '/'. An output is configured as active high when the polarities of *both* the pin name *and* equation match, i.e., both do not have the slash prefix '/'. In the examples in Figure 4-4, OUTA and OUTB are both active-low outputs. OUTC is active high. (OUTD is also active high—when slashes are used in the pin name *and* equation feeding the pin, they cancel each other out).

```
;PIN   10      11      12      13
      OUTA    /OUTB   OUTC    /OUTD

EQUATIONS

    /OUTA = A * B * C * /D    ; active low output

    OUTB = A * B * C * /D     ; active low output

    OUTC = A * B * C * /D     ; active high output

    /OUTD = A * B * C * /D    ; active high output
                              ; polarities cancel each
                              ; other
```

Figure 4-4. Active-High and Active-Low Output Equations

NOTES

Some logic compilers automatically treat outputs/equations as active low for devices with fixed, active-low outputs (e.g., a 16L8), even when the equation or signal name does not include the slash '/' prefix. Since Intel µPLDs contain programmable outputs, the presence of the slash '/' prefix is always required to designate an equation/output as active low. The absence of the prefix always means active high.

The automatic inversion option allows the parser and minimizer to reduce the number of p-terms by using DeMorgan's inversion rules. The parser and minimizer (when selected) can choose the proper combination of true/complement inputs and invert/non-invert to help designs fit into a device. This can be done automatically (Automatic Inversion = Yes) or at the designer's discretion (Automatic Inversion = No).

An alternate method for specifying an output as active high or active low is to use the "HIGH" or "LOW" keyword in the pin declaration. This option is shown below:

```
PIN  11   OUTB LOW
PIN  10   OUTA HIGH
```

Note that this option is not supported with the older order-specific method of pin declaration

Output Enable

Output enables for each macrocell on the supported devices are controlled by one or more p-terms that do not include an invert option. Equations that evaluate to a logic 1 (high) enable the output on Intel µPLDs; equations that evaluate to a logic 0 (low) disable the output (three state). (Some PLD manufacturers use a low to enable and a

```
;PIN      10      11
         OUTA    OUTB
EQUATIONS

    OUTA = B * C * /D

    /OUTB = /A * B * C * D

    OUTA.TRST = E * /G          ; output enable for OUTA

    OUTB.TRST = E * G * /D      ; output enable for OUTB
```

Figure 4-5. Outputs with Output Enable Equations

high to disable outputs.) If equations are not specified for output enables, PLDasm ties the outputs high (always enabled). Output enable equations are identified by the output signal name with a ".TRST" extension. Figure 4-5 shows two examples of output enable equations.

Registered Circuits

Registered circuits are easily implemented using the same Boolean equation syntax as combinatorial circuits, but with a ":=" operator in place of the "=". This syntax implements a standard D-type register circuit.

When using the dedicated clock pin on a µPLD with a single clock, it is not necessary to specify the clock input. PLDasm will use the dedicated clock by default. You *may*, however, choose to specify the clock as part of your design methodology. The clock for the circuit uses the output name with a ".CLKF" extension on the left-hand side of the "=" symbol and the clock input name on the right-hand side. Figure 4-6 shows several examples of registered circuits.

The first register, QOUTA, is clocked by the dedicated clock pin (since no clock is specified) and is always enabled (OE tied to VCC).

The second register, QOUTB, is also clocked by CLK. Its output buffer, however, is controlled by the OE input signal. The clock is specified here.

The third register illustrates the use of register feedback in designs. This circuit uses feedback from the first two registers as its input. It is clocked by CLK (not specified) and its output buffer is controlled by a p-term.

On devices that have more than one clock pin (e.g., the 85C060 has two clock pins), you should specify the clocks to avoid conflicts. The fourth example shows this (OUT3 and OUT13).

Some devices such as the 85C22V10 contain a programmable synchronous clock inversion option to allow registers to be clocked on either the rising or falling edge of the global clock signal. The fifth example shows outputs clocked by the non-inverted and inverted clock (OUT4 and OUT5).

The following is a list of all the signal extensions and their descriptions:

Extension	Description
.ACLK	Asynchronous Clock
.CLKF	Clock Pin (Synch.) or Clock Equation (Asynch.)
.ALE	Asynchronous Latch Enable
.LE	Synchronous Latch Enable
.TRST	Output Enable Equation
.RSTF	Clear Equation
.SETF	Preset Equation
.D	Data Input to D-Type Register
.T	Data Input to Toggle Register
.J	J Data Input to JK Register
.K	K Data Input to JK Register
.R	R Data Input to SR Register
.S	S Data Input to SR Register
.FB	Feedback Path from I/O Pin (sometimes needed during simulation)

```
; OUTPUT, CLOCK, AND OE PINS

CLK   OE   CLK2   QOUTA   /QOUTB   QOUTC  OUT4  OUT5

EQUATIONS

     ; D-register with default clock and OE always enabled

     QOUTA      := IN1 * IN3
                +  IN1 * /RESET
     QOUTA.TRST = VCC


     ; D-register with clock specified and OE controlled
     ; by an input pin (OE)

     QOUTB      := INA * /IN1
     QOUTB.CLKF = CLK
     QOUTB.TRST = OE

     ; D-register with default clock, output controlled by
     ; p-term, equation includes feedback from other
     ; registers

     /QOUTC     := QOUTA * /QOUTB
     QOUTC.TRST = IN1 * OE


     ; Two D-registers, each controlled by different clocks.
     ; Supported on devices with multiple clocks or that
     ; support asynchronous clocking.

     /QOUT3     := INA * INB
     QOUT3.CLKF = CLK

     QOUT13     := INA * /INB * IND
     QOUT13.CLKF = CLK2

     ; Two D-registers, one clocked by CLK
     ; and the other by CLK inverted.

     OUT4 := IN1 * IN4 * INA
     OURT4.CLKF = CLK

     OUT5 := IN2 * INB
     OUT5.CLKF = /CLK
```

Figure 4-6. Registered Circuits Using Boolean Equations

Using Additional Features

This section covers design of circuits that make use of the additional features provided by some Intel µPLDs. Topics include asynchronous clocking of registers, use of the Preset and/or Clear p-term, and use of T-type, JK-type, and SR-type registers. Also described is the syntax for using global resources and for taking advantage of some of the more complex I/O architectures of Intel µPLDs. The features described here are not available on all devices.

Asynchronous Clocking

Asynchronous clocking is available on many Intel µPLDs. Asynchronous clocking of registers is implemented by assigning the register clock to an equation or pin other than a dedicated clock input. This allows logic functions to be used as register clocks. For example, each register in the iPLD610 and 85C060 contains an OE/Asynchronous Clock p-term, which allows each of the 16 registers can be clocked independently.

The first example in Figure 4-7 shows a simple equation being used to clock QUTD. Note that the clock signal uses the output name with an ".ACLK" extension. A single p-term equation is then specified as the clock signal. The output buffer is tied to VCC (always enabled). Use of the register name with an ".ACLK" extension is the recommended method for specifying asynchronous clocks.

An alternate method for specifying asynchronous clocks is to use a ".CLKF" extension. Note, however, that the PLDasm compiler will assign a clock signal using this extension to a dedicated synchronous clock pin if the clock signal is: (1) driven by a single active-high input signal, and (2) the input signal is unassigned or inadvertently assigned to a pin that can function as a dedicated clock. For sake of consistency, the ".ACLK" extension is recommended.

Preset P-Term

Asynchronous Presets are available on each macrocell in the 5AC312 and 5AC324 to allow registers to be independently preset (to 1) when the specified equation is true (high). The Preset equation is implemented by using the output name with a ".SETF" extension on the left-hand side of the "=" sign and an equation on the right-hand side. The third example in Figure 4-7 shows an example of a single p-term preset equation.

Clear P-Term

An asynchronous Clear p-term is available on many devices to allow registers to be independently reset (to 0) when the specified equation is true (high). The second and fourth examples in Figure 4-7 make use of this feature. Both examples use a single p-term equation to clear a register. The Clear function is implemented in an equation by using the output name with a ".RSTF" extension on the left-hand side of the "=" sign and an equation on the right-hand side.

```
;OUTPUT , CLOCK, AND OE PINS

CLK1    OE    CLK2    QOUTD    /QOUTT    QOUTK    QOUTR  SET

EQUATIONS
; D-register with asynch. clock, OE always enabled

     QOUTD     := INA * INB
                +  INA * INC * /IND * /RESET
                +  /QOUTD * INC * INE
     QOUTD.ACLK = IN1 * IN2 * /IN4
     QOUTD.TRST = VCC

; Asynchronous clock could also be implemented as follows:
;
; QOUTD.CLKF = IN1 * IN2 * /IN4
; T-register with synch. clock, asynch. reset, OE control

     QOUTT.T   := INA * /INC
     QOUTT.CLKF = CLK1
     QOUTT.RSTF = IN1 * IN2 * RESET
     QOUTT.TRST = OE * /RESET


; JK-register with synch. clock, OE control, preset

     QOUTJ.J   := QOUTD * /INA
     QOUTJ.K   := INB * INC * /QOUTD
     QOUTJ.CLKF = CLK1
     QOUTJ.TRST = IN1 * OE
    QOUTJ.SETF = SET

; SR-register with synch. clock, asynch. reset, OE always
; enabled

     QOUTR.R   := QOUTA * IN1 * IN2 * /QOUTB
     QOUTR.S   := /QOUTA * INA * INB
                +  /QOUTB * IN1 * IN2
                +  RESET * /IND
     QOUTR.CLKF = CLK1
     QOUTR.RSTF = IN1 * IND
     QOUTR.TRST = VCC
```

Figure 4-7. Extended Register Options

Toggle Flip-Flops

A T-type, or toggle, flip-flop is available on many devices and is implemented as shown in the second example in Figure 4-7. QOUTT is the output pin name. A ".T" extension is added to the output name to designate the register input. QOUTT is clocked by a synchronous clock (dedicated clock pin) and includes an output enable p-term and a reset p-term.

JK Flip-Flops

QOUTJ is the output pin name for a JK-type flip-flop. The "J" input is identified by using the output name with a ".J" extension. In the same way, the "K" input uses the output pin name with a ".K" extension. Note that this is a clocked emulation of an RS flip-flop, not a true asynchronous circuit. (See the notes after "SR Flip-Flops".) OUTJ also includes an output enable p-term.

SR Flip-Flops

QOUTR is the output pin name for an SR-type flip-flop. The "S" input is identified by using the output pin name with a ".S" extension. In the same way, the "R" input uses the output pin name with a ".R" extension. Note that this is a clocked emulation of an RS flip-flop, not a true asynchronous circuit. QOUTR also includes a clear p-term and the output buffer is tied to VCC (always enabled).

NOTES

Intel μPLDs emulate JK and SR flip-flips as synchronous registers. The registers do not change state until the clock signal changes.

When JK and SR flip-flops are selected, the product terms are shared among two OR gates (one OR gate for the J or R input, and one OR gate for the K or S input). The allocation of these product terms may be described as follows:

J or R = n
K or S = (available terms – n)

Using Global Set/Reset Signals

Some devices contain global signals, such as a register set and a register reset. The iPLD22V10, 85C22V10, and 5C031, for example, both contain a synchronous set p-term and an asynchronous reset p-term. The set and reset signals can be driven by the true or complement of any input or I/O pin, or by any logic equation that can be implemented in a single AND expression. Figure 4-8 shows an example of these global signals, along with the PLDasm syntax to make use of them.

Note that a virtual pin (pin 25 on the iPLD22V10 and 85C22V10 and pin 21 on the 5C031) is used to identify the global signal, "GLOBAL" in this case. This name is then used in the equations section, along with the ".SETF" and ".RSTF" extension to specify the logic for the set and reset signals, respectively. In the example, the synchronous set is connected to the SET input (pin 2). All registers in the device will be set (preset) to a logic one when SET and ENABLE are high and the registers are clocked. The synchronous reset is connected to the complement of the RESET signal (pin 3). All registers in the device will be reset (cleared) to zero when RESET goes low and ENABLE goes

high. Note that the programmable inverter in the iPLD22V10, 85C22V10, and 5C031 is located at the output of the register. When using register set and reset to set the register ouput, the inverter in these devices (when used) will invert the output of the register from the internal high or low state.

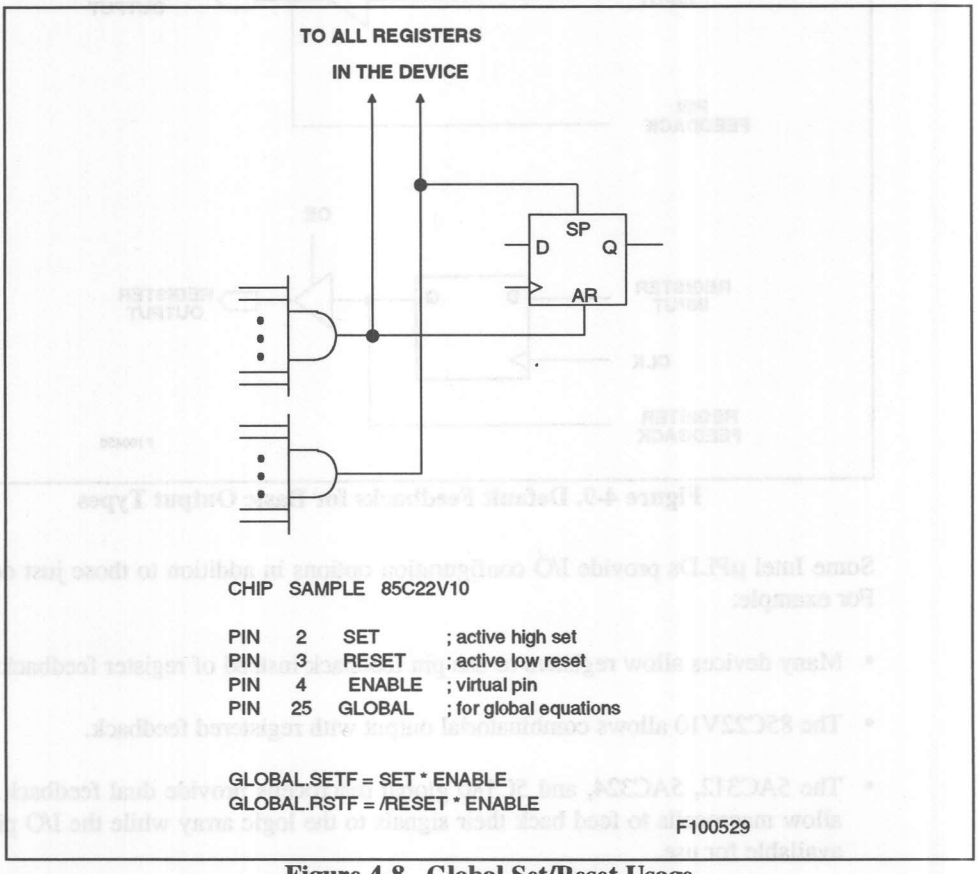


Figure 4-8. Global Set/Reset Usage

Implementing Alternate I/O Options

The default I/O configurations for Intel µPLDs are shown in Figure 4-9. For combinatorial outputs, the default feedback option is "pin feedback". This means that the feedback connection to the logic array comes from the I/O pin (after the output buffer). For registered outputs, the default feedback option is "registered feedback". This means that the feedback connection to the logic array comes from the register output (before the output buffer). These defaults do not need to be specified in the PLDasm file (although you may desire to do so as part of your design methodology).

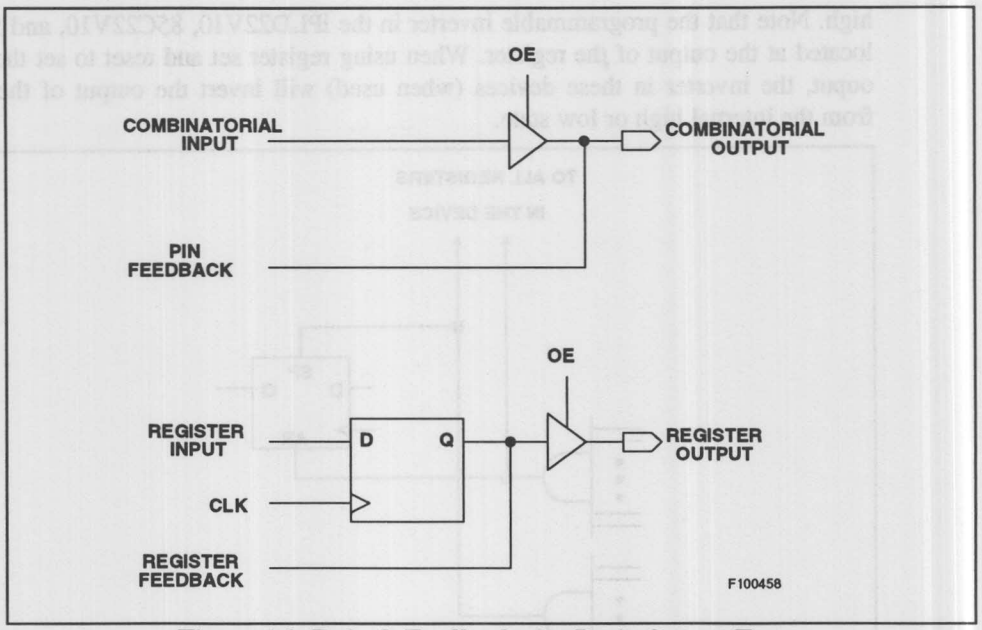


Figure 4-9. Default Feedbacks for Basic Output Types

Some Intel µPLDs provide I/O configuration options in addition to those just described. For example:

- Many devices allow registers to use pin feedback instead of register feedback.
- The 85C22V10 allows combinatorial output with registered feedback.
- The 5AC312, 5AC324, and 5C180 global macrocells provide dual feedback paths to allow macrocells to feed back their signals to the logic array while the I/O pin is still available for use.

To specify an alternate I/O configuration, use the appropriate keyword(s) in the pin declaration for I/O signals to be configured (see Figure 4-10). The keywords are not order-dependent, but you may wish to adhere to an order as part of your design methodology. If a keyword and the equation operator for an output conflict, the keyword takes precedence.

A complete list of all I/O keywords is shown on the facing page. The keywords designated by an asterisk are not available on supported devices.

I/O Keyword	Description
COMBINATORIAL or COMB	Combinatorial Output
REGISTERED or REG	Registered Output
LATCHED	Latched Output
PINFBK	Pin Feedback
CMBFBK	Combinatorial Feedback
REGFBK	Registered Feedback
*LATFBK	Latched Feedback
BURIED	Internal Feedback
HIGH	Active-High Output
LOW	Active-Low Output
INPUT	Input Pin
OUTPUT	Output Pin
I/O	I/O Pin

*Not available on supported devices

```
; Default specifications - No keyword. Will be combina-
; torial or registered as determined by equation operator.


     PIN       5    COUT1
     PIN       6    ROUT1


; Combinatorial Output, Pin Feedback. Same as combinatorial
; default, but this time it is specified using keywords.

     PIN       5    COUT1     COMBINATORIAL          PINFBK

; Registered Output, Registered Feedback. Same as
; registered default, but this time it is specified
; using keywords.

     PIN       6    ROUT1     REGISTERED             REGFBK


; Registered Output, Pin Feedback.

     PIN       7    ROUT2     REGISTERED             PINFBK


; Combinatorial Output, Registered Feedback.

     PIN       15   CORF3     COMB       REGFBK
```

Figure 4-10. Examples of I/O Options Using PLDasm Syntax

The following pages show how to take advantage of these architectural features in PLDasm syntax. Note that the "INPUT", "OUTPUT", and "I/O" keywords are supported for compatibility with PALASM-2 source files only. The PLDasm compiler accepts these keywords but does not process files differently than when they are omitted.

Automatic Pin Assignments

Automatic pin assignment can be implemented by omitting the pin *numbers* in the declaration section of the source file. Since the device and package are known to PLDasm, the PLDasm fitter uses its internal algorithms to fit designs to the target devices. This helps designers concentrate on implementing the desired functions by reducing the need to know device pinout.

While the fitter does not require that pin numbers be assigned, it does require that enough information be present in the design to remove ambiguities. The recommended approach is to specify all control signals for all equations in the design, including dedicated clock pins. For example, the output equation below contains equations for the OE, Set, Reset, and Synchronous Clock control signals in addition to the normal SOP equation.

```
REG1 := IN1 * IN2 * /IN3 * /IN4 * ENA
     + IN1 * /IN2 * REG2 * ENA
     + /IN1 * /REG3 * /IN4

REG1.TRST = OE
REG1.CLKF = CLK20MHZ
REG1.RSTF = /RESET
REG1.SETF = SET
```

If all outputs in a design have the control signals specified as shown above, the PLDasm fitter can provide the pin assignments. In these cases, the Pin Declarations can be written without pin numbers, as follows:

```
PIN      CLK20MHZ
PIN      OE
PIN      RESET
PIN      SET
PIN      ENA
PIN      IN1
PIN      IN2
PIN      IN3
PIN      IN4
PIN      REG1
PIN      REG2
PIN      REG3
```

When this guideline is followed, PLDasm source files can be compiled to any device that supports the requested feature set. The PLDasm fitter will supply the appropriate pin numbers for the input and I/O signals.

An alternative to specifying dedicated clock signals, is to assign the pins for the output signals. The PLDasm fitter can then assign the outputs to the appropriate clock signal,

even on devices that support multiple clock pins. The following example repeats the pin assignments and equations from the previous example, showing this alternate form.

```
PIN      CLK20MHZ
PIN      OE
PIN      RESET
PIN      SET
PIN      ENA
PIN      IN1
PIN      IN2
PIN      IN3
PIN      IN4
PIN  16  REG1
PIN  17  REG2
PIN  18  REG3

REG1 := IN1 * IN2 * /IN3 * /IN4 * ENA
     + IN1 * /IN2 * REG2 * ENA
     + /IN1 * /REG3 * /IN4

REG1.TRST = OE
REG1.RSTF = /RESET
REG1.SETF = SET
```

Note that because Intel µPLDs do not contain dedicated OE, Set, or Reset pins, all other control signals must still be specified.

Bidirectional I/O

Bidirectional I/O can be implemented on all Intel µPLDs that support pin feedback. The output enable to these pins determine whether the pin is an output or an input (see Figure 4-11 for an example). When the XMT_RCV p-term is active (high), the macrocell drives both the output pin and the input feedback signal. When XMT_RCV is low, any external device can drive the input feedback signal; the I/O pin acts as an input. Note that it is not necessary to list the bidirectional input signal in the input pin declaration to use this feature.

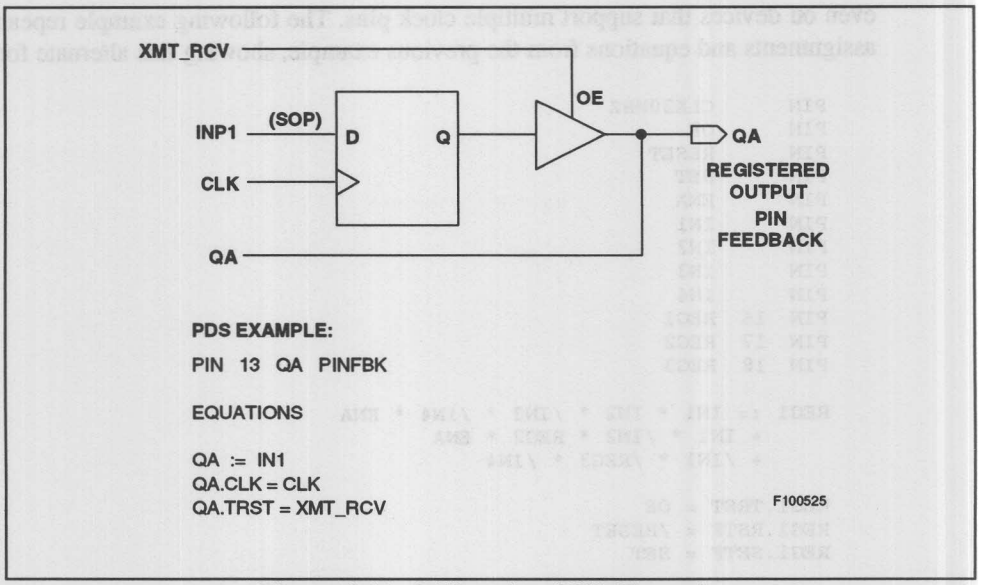


Figure 4-11. Bidirectional I/O Example

Dual Feedback/Buried Macrocells

Some µPLDs, such as the 5AC312/5AC324, and the global macrocells on the 5C180, contain dual-feedback paths. Dual feedback allows the associated I/O pin of a macrocell to be used for input (pin feedback). As shown in Figure 4-12, a second (internal) feedback can be taken from the output of a register. The OE for the macrocell is disabled, allowing the I/O pin to function as an input and the now-buried register output to be used as well. Note that pin 15 is listed twice in the pin declarations, once for an input and once for an output.

Dual Feedback with Bidirectional I/O

Dual feedback with bidirectional I/O can be used with devices such as the 5AC312 and 5AC324 and the global macrocells of the 5C180. This is similar to the simple dual-feedback example shown in Figure 4-12. In Figure 4-13, instead of the OE p-term being disabled, it is used to allow the I/O pin to function as both input and output. This capability allows the register output to be used internally regardless of the state of the output buffer. The signal on the I/O pin will depend on whether the pin is transmitting or receiving data.

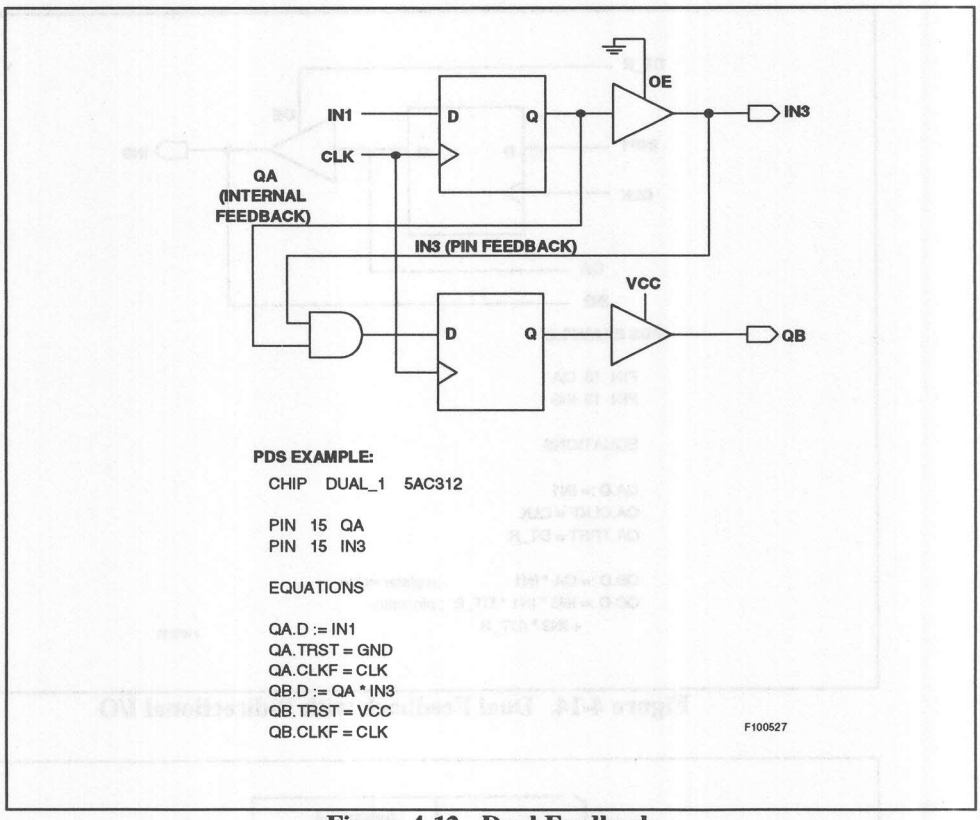


Figure 4-12. Dual Feedback

P-Term Allocation

Some Intel µPLDs, such as the 5AC312 and 5AC324, allow p-terms from one macrocell to be allocated to other macrocells. This can give you from zero to 16 p-terms for your equations. Figure 4-14 shows an example where Macrocell 3 is using 16 p-terms, Macrocell 4 is using 4 p-terms, Macrocell 5 is using 12 p-terms. P-terms are allocated in blocks of four, so not all p-terms in a block may be used.

P-term allocation is performed automatically by the fitter. Designers take advantage of this feature by simply creating equations that meet their design requirements. The minimizer will reduce the number of p-terms and the fitter will generate the correct JEDEC image to allocate the needed p-terms.

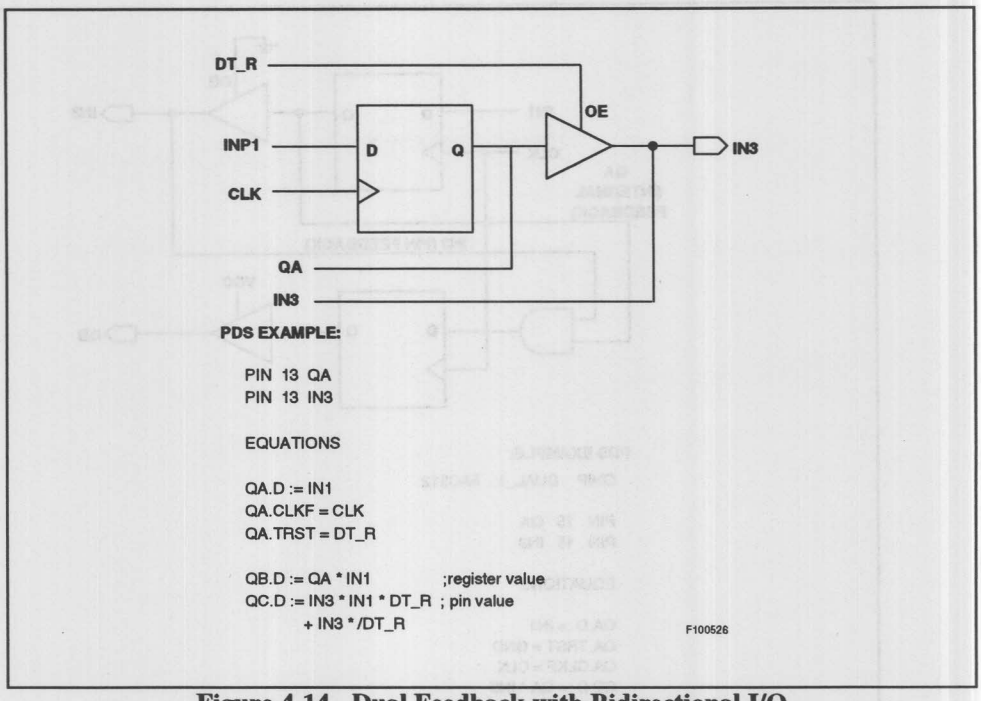


Figure 4-14. Dual Feedback with Bidirectional I/O

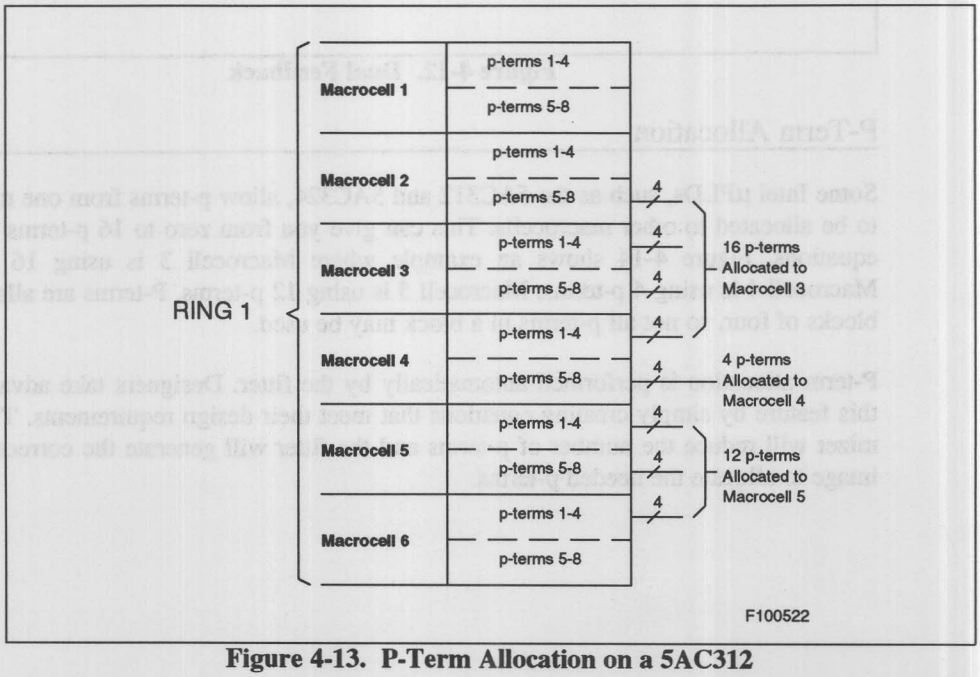


Figure 4-13. P-Term Allocation on a 5AC312

Using Disconnected Macrocells

In devices with p-term allocation, such as the 5AC312/5AC324, inputs to a macrocell are disconnected from the logic array when both p-term groups are allocated to adjacent macrocells. If all the inputs to a macrocell are disconnected, the macrocell input can be tied to VCC or GND (see Figure 4-15).

Also, the control signals, Output Enable, Preset, Clear, and Synchronous or Asynchronous Clock, are still available which allows the macrocell to be used to implement a variety of useful functions, such as:

- Toggles using T-register and D-register equations
- Asynchronous RS registers driven by preset and clear p-terms
- Other registered functions that use the control signals

The ability to use disconnected registers can also be combined with feedback and dual-feedback support to make the most efficient use of device resources.

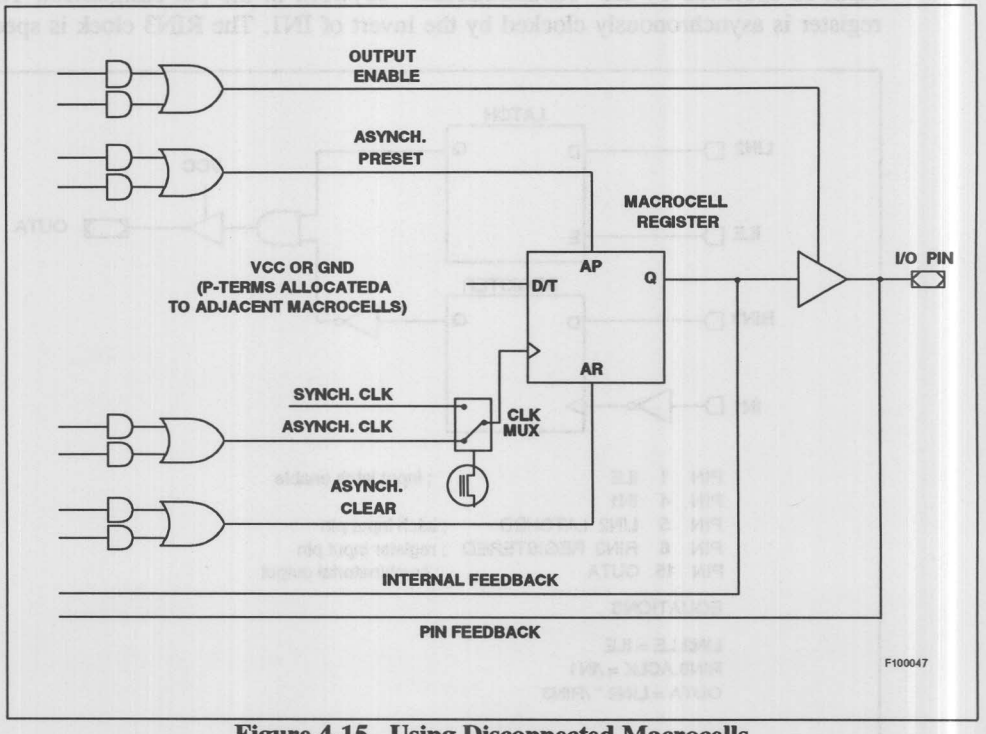


Figure 4-15. Using Disconnected Macrocells

The 5AC312 and 5AC324 have programmable inputs that can be individually configured in one of five modes:

1. Flow-through input
2. Input register (D-register), synchronous operation
3. Input register (D-register), asynchronous operation
4. Input latch (transparent D-latch), synchronous operation
5. Input latch (transparent D-latch), asynchronous operation

Figure 4-16 shows an example of a flow-through, a synchronous latch , and an asynchronous register with PLDasm syntax examples. IN1 is a standard flow-through input. LIN2 is a latched input as specified by the "LATCHED" keyword in the pin assignment. The latch enable for this input is specified as ILE (synchronous input latch/clock) in the equations section by using the signal name with a ".LE" extension. RIN3 is a registered input as specified by the "REGISTERED" keyword in the pin assignment. This input register is asynchronously clocked by the invert of IN1. The RIN3 clock is specified as

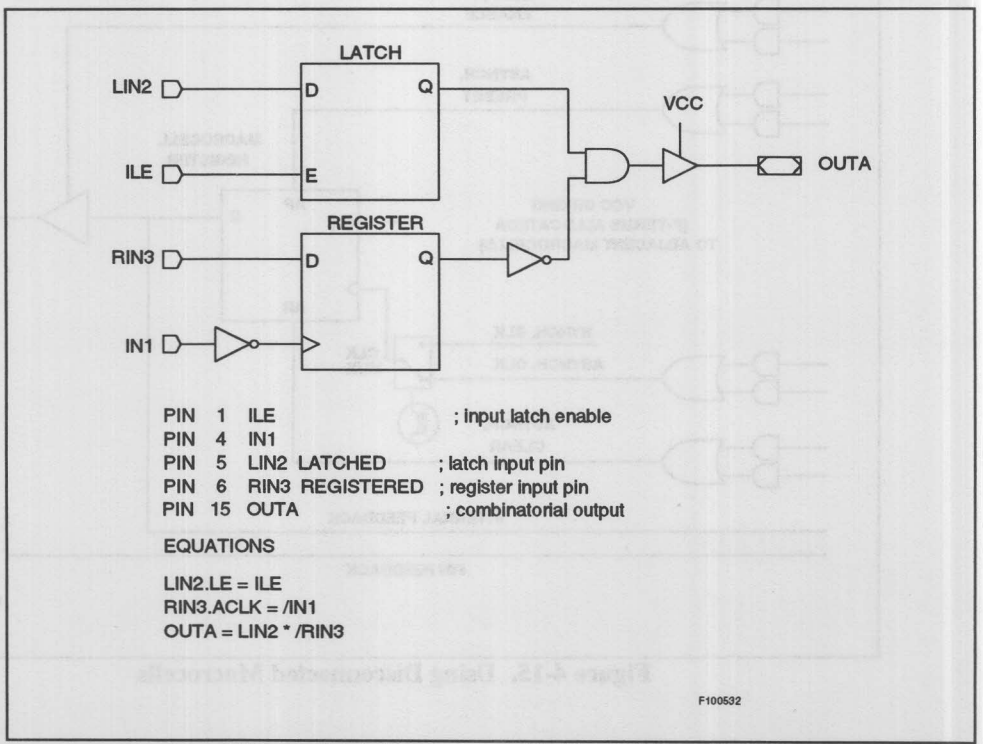


Figure 4-16. Programmable Input Examples

an asynchronous clock by using the ".ACLK" extension. OUTA is shown as a simple example to allow a simulation section to be written to test these examples.

When clocking latches/registers asynchronously (i.e., using a p-term), the clock can be generated by the true or complement of any pin, or by any p-term that can be expressed in a single AND form.

When using the synchronous ILE/ICLK clock pin, input latches open (become transparent) on the logic high and latch data on the falling edge of the signal. Input registers clock on the falling edge of the clock signal. (Note that this is different from macrocell registers, which clock on the rising edge of the clock signal.)

The ability to latch/clock data into the device inputs can help implement fast pipelining circuits.

Altering Set-Up and Hold Times

On some µPLDs, such as the iPLD610, 85C060, iPLD910, and 85C090, there is a way to alter the t_{SU} (setup) and t_{CO} (clock-to-output) parameters to better match system timing. By using the asynchronous clocking feature provided with these devices, the t_{SU} and t_{CO} values are shifted with respect to the signal or signals clocking the registers. For example, using the asynchronous clock on the iPLD610 shortens t_{SU} and lengthens t_{CO}. The following table shows the timing differences for specific speed versions of the 85C060 and 85C090 mPLDs. Please refer to the device data sheets for additional parts and speed versions.

Device	Synchronous Clocking		Asynchronous Clocking	
	t_{su} (ns)	t_{co} (ns)	t_{su} (ns)	t_{co} (ns)
iPLD610-10	7.0	6.5	2.0	12
85C060-10	7.0	6.5	2.0	12
iPLD910-15	11	9.0	4.0	18
85C090-15	11	9.0	4.0	18

The following are PLDasm examples of synchronous and asynchronous clocking for the iPLD610, 85C060, iPLD910, and 85C090:

Synchronous Clocking

```
PIN   1   CLK          ; synch. 610/910 clock
PIN  10   OUT_S

EQUATIONS

OUT_S.CLKF = CLK
```

Asynchronous Clocking

```
PIN   2   ASYNCLK          ; any input or I/O pin
PIN  10   OUT_S

EQUATIONS

OUT_S.ACLK = ASYNCLK
```

An alternate method of tailoring t_{SU} and t_{SU} to system timing is provided on the 85C22V10 in the form of a clock inversion option. Use of this feature allows selected registers to clock on the falling edge of a synchronous clock signal, effectively moving the t_{SU} and t_{SU} windows with respect to the rest of the system timing. Refer to the discussion of "Registered Circuits" earlier in this chapter for an example.

Truth Table Design

Truth tables provide an effective way of describing a design or parts of a design. For example, truth tables are often used for decoders. PLDasm supports more than one optional Truth Table section in PLDasm files.

As shown in Figure 4-17, each Truth Table section begins with the "T_TAB" keyword. Truth tables are position dependent, i.e., each input has a corresponding column for each row of the table. The first line of the table lists the input and output signal names for the Truth table inside parentheses. Subsequent rows list the values of each output for each combination of inputs.

```
; Combinatorial truth table

T_TAB  ( in1    in2   >>   out1   out2  /out3 )
           0      0     :    0      0     0
           0      1     :    0      0     1
           1      0     :    1      0     0
           1      1     :    1      1     0


; Registered truth table

T_TAB ( UPDOWN  /CLR   q0   q1  :>>  q0.T  q1.D )
           1      0     0    1   :    0     0
           1      0     1    0   :    0     0
           0      0     0    1   :    1     1
           0      0     1    1   :    1     0
           X      0     0    x   :    1     1
```

Figure 4-17. Truth Table Examples

Inputs are listed first. These are followed by a separator. Outputs are then listed. Entries can be separated by blank spaces or commas. The separator in the signal name line also identifies truth table outputs as combinatorial (>>), registered (:>>), or latched (*>>). The sample file ADDR1.PDS in your installation directory shows a truth table with latched outputs.

All outputs in the same truth table must be the same type (i.e., you cannot mix combinatorial and registered outputs in the same truth table). The first table in Figure 4-17 is combinatorial, the second is registered. The separator for all other lines in the table is a colon (:).

Legal values for truth table entries are:

0 = False
1 = True
X = Don't Care

Don't Cares may be used to describe input or output values. A Don't Care for an output is treated as a logic 0.

Inputs or outputs can be specified as true or complemented. For example, "out2" is true and "/out3" is complemented. Thus "/out3" goes high for cases where the truth table entry specifies a 0; it goes low where the truth table entry specifies a 1.

For truth tables using registered outputs, the register type can be selected by adding the appropriate extension to the output signal name In the second truth table, q0.T specifies a T-type register and q1.D specified a D-type register. This feature can be useful when it is known that a particular type will end up using fewer p-terms, but only when using devices that support T-type registers.

Truth table outputs can only drive output pins directly. They can be used as inputs to Boolean equations, state machines, or other truth tables indirectly, i.e., via feedback paths to the logic array. All truth table outputs must be specified in the pin list.

Truth table inputs can come from input or I/O pins. This means that Boolean equations, state machines, or other truth tables can provide input to truth tables via feedback paths to the logic array.

A Truth Table section ends with the next PLDasm keyword.

State Machine Design

State machines provide an effective way of describing sequential (registered) logic. A designer typically draws a state diagram that represents the different states and transitions for a design. This diagram can then be expressed in PLDasm's state machine syntax. PLDasm supports more than one optional State Machine section in PLDasm files.

Each state machine begins with the "STATE" keyword followed by a machine type ("MEALY_MACHINE or MOORE_MACHINE"). Outputs on Moore machines depend on the current state only. Outputs on Mealy machines depend on both the current state and next state information. This is followed by subsections that identify the global defaults, state transitions, output values, and transition conditions. Synchronous state machines transition on the rising edge of a dedicated clock pin. Asynchronous state machines transition when the specified condition used as the clock is true.

NOTE

PLDasm supports asynchronous state machines, but does not perform hazard checking. You should check the resulting equations for hazard conditions to ensure that your design will function properly.

Simple Moore State Machine Example

Figure 4-18 is an example State Machine section that implements a 2-bit up/down counter (2BIT.PDS in your installation directory). The counter is a Moore machine defined to have four states (S1 through S4), with state S1 as the default state. The state assignments subsection defines the output values for each state. As can be seen, the

```
CHIP  2_bit_count  85C220

; pins

PIN  1    CLK
PIN  2    UPDOWN
PIN  3    CLEAR
PIN  12   Q1
PIN  13   Q0

; Simple 2-Bit State Machine

STATE
MOORE_MACHINE
DEFAULT_BRANCH S1

; state assignments

     S1 = /Q1 * /Q0
     S2 = /Q1 *  Q0
     S3 =  Q1 * /Q0
     S4 =  Q1 *  Q0

; state transitions

     S1 := UP       -> S2
         + DOWN     -> S4

     S2 := UP       -> S3
         + DOWN     -> S1

     S3 := UP       -> S4
         + DOWN     -> S2

     S4 := UP       -> S1
         + DOWN     -> S3

; input conditions transitions

CONDITIONS

     UP = UPDOWN * /CLEAR
     DOWN = /UPDOWN * /CLEAR
```

Figure 4-18. 2-Bit Counter State Machine

outputs follow a binary sequence. The state transitions subsection defines the order of transitions from one state to another based on transition conditions.

The conditions subsection is denoted by the "CONDITIONS" keyword. In the figure, UP and DOWN are mutually exclusive, with both qualified by CLEAR. This means that when CLEAR is asserted (high), the default branch to S1 is taken, which clears the counter to 00. When CLEAR is not asserted (low) the state machine counts up when UPDOWN is high and counts down when UPDOWN is low. (Input conditions can only be combinatorial equations. They can be specified in the Conditions subsection or in the transitions section. When specified in the transitions section, they should be enclosed in parenthesis, as follows:

```
S7 := BUS_CONT                -> S8     ; S4 if ready
    + (/RDY * RESET * B2)     -> S10    ; stay if not ready
```

State Machine Format (Moore Machine)

This section of the guide describes state machine format for PLDasm files in greater detail. In Figure 4-19, the "STATE" keyword and the machine type keyword are followed by five subsections: (1) Machine Defaults, (2) State Assignments, (3) State Transitions, (4) Transition Outputs, and (5) Transition Conditions. (A Mealy State Machine appears later in this section.)

Machine Defaults

The Machine Defaults subsection can include three defaults: Output Hold, Default Output, and Default Branch. Each State Machine section can use one, two, or all three defaults. When using more than one default, they must appear in the order shown.

The "OUTPUT_HOLD" keyword, followed by a list of output pins, specifies that those outputs are to hold their previous state if it is not possible to determine which state to transition to. Use of this option prevents outputs from changing states when all possible state transitions are not specified. Figure 4-19 uses this default.

The "DEFAULT_OUTPUT" keyword, followed by a list of output pin names, defines the default levels for those outputs if it is not possible to resolve a transition. Use of this option prevents outputs from going to unknown states when all possible state transitions are not specified. Should an unspecified state occur, the outputs will be driven to the defined levels. Valid logic levels are low (/ prefix), high (no prefix), or don't care (% prefix).

The "DEFAULT_BRANCH" keyword, followed by the name of a state or additional keyword, defines the default state to which the machine will transition if it is not possible to decode input conditions. Use of this option eliminates the need to specify all possible combinations of input conditions. Should an unspecified input combination occur, the machine will behave as specified. Three behaviors are possible:

1. Transition to the default state. This is specified by including the name of the default state after the keyword, as follows:

```
DEFAULT_BRANCH    S1
```

2. Hold the current state. This is specified by including the keyword "HOLD_STATE" after the "DEFAULT_BRANCH" keyword, as follows:

```
DEFAULT_BRANCH    HOLD_STATE
```

```
Title           Local Bus Controller Example
Pattern         pds
Revision        1
Author          Your Name
Company         Your Company
Date            Date

CHIP    BUS_CON1    85C224

; inputs
PIN  1     CLK
PIN  2     M_IO
PIN  3     INT_CYC
PIN  4     A20
PIN  5     READY

; outputs
PIN  15    LOCAL
PIN  16    MEMORY
PIN  17    INTACK
PIN  18    Q3
PIN  19    Q2
PIN  20    Q1
PIN  21    Q0
; simple bus controller
; inputs are M_IO, INT_CYC, A20, and READY
; outputs are LOCAL, MEMORY, and INTACK
STATE
MOORE_MACHINE
; define defaults

OUTPUT_HOLD  LOCAL  MEMORY  INTACK

DEFAULT_BRANCH  S0
; state assignments

     S0 = /Q3 * /Q2 * /Q1 * /Q0
     S1 = /Q3 * /Q2 * /Q1 *  Q0
     S2 = /Q3 * /Q2 *  Q1 * /Q0
     S3 = /Q3 * /Q2 *  Q1 *  Q0
     S4 = /Q3 *  Q2 * /Q1 * /Q0
     S5 = /Q3 *  Q2 * /Q1 *  Q0
     S6 = /Q3 *  Q2 *  Q1 * /Q0
     S7 = /Q3 *  Q2 *  Q1 *  Q0
     S8 =  Q3 * /Q2 * /Q1 * /Q0
```

Figure 4-19. Example State Machine Section

```
; state transitions

S0 := MEM_REQ  -> S1          ; S1 on local memory request
   + INT_REQ   -> S3          ; S3 on interrupt request
                              ; no-op (S0) if no request
; memory cycles

S1  :=  VCC          -> S2          ; one fixed wait cycle

S2  :=  BUS_CONT     -> S7          ; S7 if ready
     +  BUS_WAIT     -> S2          ; stay if not ready

; interrupt cycles

S3  :=  BUS_CONT     -> S4          ; S4 if ready
     +  BUS_WAIT     -> S2          ; stay if not ready

S4  :=  VCC          -> S5          ; jump to S5 - fixed wait
S5  :=  VCC          -> S6          ; jump to S6 - fixed wait

S6  :=  BUS_CONT     -> S7          ; jump to S7 if interrupt done
     +  BUS_WAIT     -> S6          ; stay if not done

; cleanup and idle cycles

S7  :=  VCC          -> S8          ; cleanup, then jump to S8
S8  :=  VCC          -> S0          ; jump to S0, ready to start

; transition outputs

S0.OUTF  :=   LOCAL * /MEMORY * /INTACK

S1.OUTF :=  /LOCAL *  MEMORY * /INTACK
S2.OUTF :=  /LOCAL *  MEMORY * /INTACK
S3.OUTF :=  /LOCAL * /MEMORY *  INTACK

S4.OUTF :=  /LOCAL * /MEMORY * /INTACK
S5.OUTF :=  /LOCAL * /MEMORY * /INTACK
S6.OUTF :=  /LOCAL * /MEMORY *  INTACK

S7.OUTF :=  /LOCAL * /MEMORY * /INTACK
S8.OUTF :=   LOCAL * /MEMORY * /INTACK

CONDITIONS

    MEM_REQ = M_IO * /INT_CYC * /A20
    INT_REQ = /M_IO * INT_CYC * /A20

    BUS_CONT = /READY
    BUS_WAIT = READY
```

Figure 4-19. Example State Machine Section (Continued)

3. Transition to the next state. This is specified by including the "NEXT_STATE" keyword after the "DEFAULT_BRANCH" keyword. The next state is defined by the order of the state assignment equations. This option is implemented as follows:

```
DEFAULT_BRANCH    NEXT_STATE
```

State Assignments

The State Assignments subsection defines each of the states in the machine, the state variables, and the values of those variables. States are defined by a state name, i.e., "S1", "S2", etc., on the left-hand side of an equal sign "=". State variables and the values of those variables are listed on the right-hand side of the equal sign. Each state must have a unique set of values for the state variables. When defining values for state variables, the choices are true (no prefix) or false (/ prefix).

State variables can have from 1-14 alphanumeric characters or underscore; the first character must be an alphabetic character.

Three possible methods for state assignment may be used. These methods are described below.

Binary State Assignment

Binary is the simplest state assignment method. Each state in the list of state assignments is encoded with the binary representation. The state machine shown below shows an example of binary state assignment. This is also the method used in Figure 4-19.

```
STATE
MOORE_MACHINE
DEFAULT_BRANCH S1

; state assignments
    S1 = /Q1 * /Q0
    S2 = /Q1 *  Q0
    S3 =  Q1 * /Q0
    S4 =  Q1 *  Q0
```

Gray Code State Assignment

A Gray code is defined as two consecutive states that differ in state assignment values by exactly one bit. The example below shows assignment of Gray codes for an 8-state machine.

```
STATE
MOORE_MACHINE
DEFAULT_BRANCH S0

;state assignments
    S0 = /Q2 * /Q1 * /Q0      ; state 0, powerup
    S1 = /Q2 * /Q1 *  Q0
    S2 = /Q2 *  Q1 *  Q0
    S3 = /Q2 *  Q1 * /Q0
    S4 =  Q2 *  Q1 * /Q0
    S5 =  Q2 *  Q1 *  Q0
    S6 =  Q2 * /Q1 *  Q0
    S7 =  Q2 * /Q1 * /Q0      ; state 7
```

One Hot State Assignment

With a One Hot code, there are generally as many output bits as there are states in the machine. Each bit is 0 except for the state that the machine is in. This is primarily useful for very small machines, or for those where the state values are driving other logic. An example of One Hot assignment is shown below:

```
STATE
MEALY_MACHINE
DEFAULT_BRANCH S0

;state assignments
    INITIAL = /Q3 * /Q2 * /Q1 * /Q0     ; state 0, powerup
         S1 = /Q3 * /Q2 * /Q1 *  Q0
         S2 = /Q3 * /Q2 *  Q1 * /Q0
         S3 = /Q3 *  Q2 * /Q1 * /Q0
         S4 =  Q3 * /Q2 * /Q1 * /Q0
```

State Transitions

The State Transitions subsection specifies the transitions between states. The subsection begins with the name of a state. State transitions that are not explicitly defined in this section will transition to the DEFAULT_BRANCH state, if any.

Each entry in this subsection contains the name of the current state, the Boolean register operator (:=) or combinatorial operation (=), a condition label, the transition designator (->), and the name of the target state. For example, the first entry in the State Transitions subsection in Figure 4-19 is as follows:

```
S0 := MEM_REQ      ->   S1        ; S1 on local memory request
    + INT_REQ      ->   S3        ; S3 on interrupt request
                                  ; no-op (S0) if no request
```

This entry means, "When you are at state S0 and MEM_REQ is true, transition to state S1 on the next clock edge; if INT_REQ is true, transition to state S3 on the next clock edge." Since S0 was previously defined as the Default Branch state, if MEM_REQ or INT_REQ are not true, the machine would remain in state S0.

The second entry is as follows:

```
S1 := VCC          ->   S2          ; on fixed wait cycle
```

Since no condition is specified, this is an *unconditional transition.* The machine will be in state S1 for one clock cycle before transitioning to state S2. If the Default Branch had been defined as "NEXT_STATE", it would not have been necessary to specify each unconditional transition; they would be assumed.

The third entry is as follows:

```
S2 := BUS_CONT     -> S4          ; S4 if ready
    + BUS_WAIT     -> S2          ; stay if not ready
```

This entry means, "When you are at state S2 and BUS_CONT is true, transition to state S4. If BUS_WAIT is true and BUS_CONT is not true remain in state S2". S0 is still the default state if neither BUS_CONT or BUS_WAIT is true; this case, however, will never occur because BUS_WAIT is defined in the Conditions section to be the complement of BUS_CONT. Conditions for a given state transition must be mutually exclusive, but do not have to be the complement of each other.

For asynchronous state machines, use the combinatorial operator (=) for state transitions, as follows:

```
S8 = WAIT_DONE     -> S9          ; jump on wait_done high
```

NOTE

PLDasm supports asynchronous state machines, but does not perform hazard checking. You should check the resulting equations for hazard conditions to ensure that your design will function properly.

Transition Outputs

The Transition Outputs subsection specifies the state of output signals during transitions for state machines where outputs are not the state registers themselves. In this case, the output signals are LOCAL, MEMORY, and INTACK while the state registers are Q0 through Q3.

Note that LOCAL, MEMORY, and INTACK have previously been defined in the OUTPUT_HOLD specification. This means that they will hold their previous state if it is not possible to resolve a transition to the next state.

The S0.OUTF entry in Figure 4-19 means, "If you are transitioning from S0, LOCAL is driven high and MEMORY and INTACK are driven low."

The entry for S1.OUTF in Figure 4-19 means, "If you are transitioning from S1, LOCAL and INTACK are driven low and MEMORY is driven high.

Another way to specify default output transitions is as follows:

```
P1.OUTF := CONDITION_1        -> /SIGNALA *  SIGNALB
         + CONDITION_2        ->  SIGNALA * /SIGNALB
         +->                      /SIGNALA * /SIGNALB
```

where the "+->" says "transition the outputs as follows if the above conditions are not met." This defines default output transitions for a particular state based on conditions. There may be only one such definition for each state. Note that the *fact that conditions are added to* the output transitions that defines these outputs as Mealy outputs (refer to the next example). When no conditions are present, the outputs are Moore outputs.

Transition Conditions

The Transition Conditions section begins with the "CONDITIONS" keyword and specifies the combination of input signals that determine the transition conditions.

For example, BUSREQ is a transition condition that is true when (1) M_IO is high and INTCYC and A20 are low or when (2) M_IO and A20 are low and INTCYC is high.

Transition conditions entries are combinatorial Boolean equations. Register/latch equations cannot be used to specify transition conditions.

Mealy State Machine Example

Figure 4-20 is a state diagram for a Mealy Machine status checker. This machine transitions between three states. The outputs of the machine, however, may set be to different values for a given state, depending on input conditions. A Moore Machine implementa-

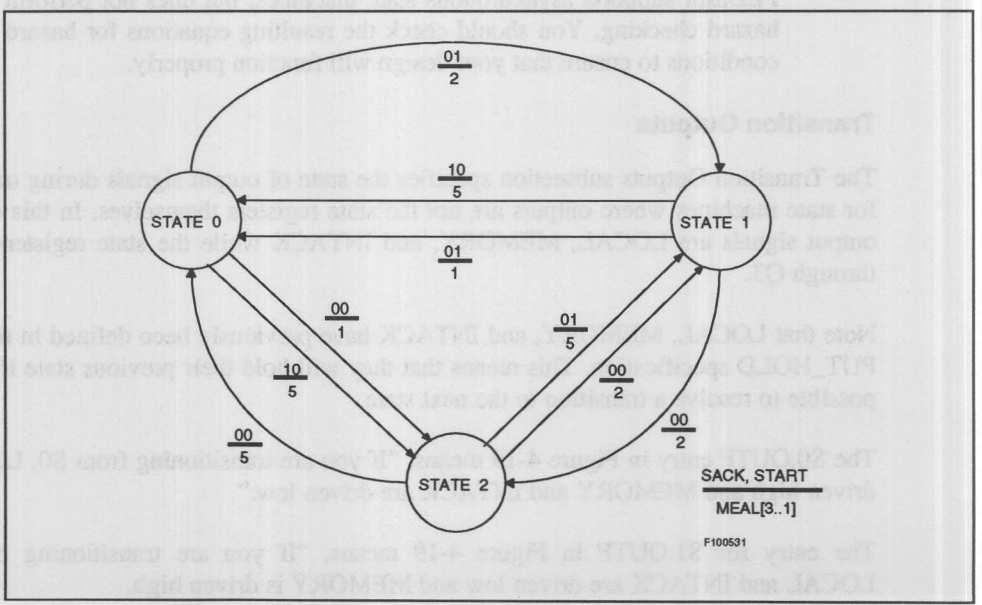


Figure 4-20. State Diagram for Mealy Status Checker

tion of this design would require nine states to accomplish the same task (one state for each possible combination of outputs). Figure 4-21 is a PLDasm listing of the design.

```
Title           Shows Mealy Machine
Pattern         pds
Revision
Author          Intel Applications
Company         Intel
Date            August 1991

CHIP exmealy1 85C224
;
; Design is a system state tracker.
;

PIN       CLOCK     ; clock for state variables
PIN       SACK      ; input conditions
PIN       START

PIN       MEAL3     ;  Mealy-machine outputs
PIN       MEAL2
PIN       MEAL1

PIN       MSTATE1   ;  Mealy state variables
PIN       MSTATE0

STRING M1_OUT '/MEAL3 * /MEAL2 *  MEAL1'
STRING M2_OUT '/MEAL3 *  MEAL2 * /MEAL1'
STRING M5_OUT ' MEAL3 * /MEAL2 *  MEAL1'

STATE MEALY_MACHINE

DEFAULT_OUTPUT  /MEAL3 /MEAL2 /MEAL1

DEFAULT_BRANCH  HOLD_STATE

; state assignments

    STATE0 = /MSTATE1 * /MSTATE0      ; powerup state
    STATE1 = /MSTATE1 *  MSTATE0
    STATE2 =  MSTATE1 * /MSTATE0

; state transitions, which state to go to next

STATE0 := TRIG0           -> STATE2
        + TRIG1           -> STATE1
        + TRIG2           -> STATE2

STATE1 := TRIG0           -> STATE2
        + TRIG1           -> STATE0
        + TRIG2           -> STATE0

STATE2 := TRIG0           -> STATE0
        + TRIG1           -> STATE1
        + TRIG2           -> STATE1
```

Figure 4-21. Mealy State Machine Example

```
; output transitions, what the output variables should be

STATE0.OUTF = TRIG0       -> M1_OUT
            + TRIG1       -> M2_OUT
            + TRIG2       -> M5_OUT

STATE1.OUTF = TRIG0       -> M2_OUT
            + TRIG1       -> M1_OUT
            + TRIG2       -> M2_OUT

STATE2.OUTF = TRIG0       -> M5_OUT
            + TRIG1       -> M5_OUT
            + TRIG2       -> M1_OUT

; conditions that determine transitions

CONDITIONS
    TRIG0 = /SACK * /START
    TRIG1 = /SACK *  START
    TRIG2 =  SACK * /START

EQUATIONS

MSTATE0.CLKF = CLOCK          ; hook clock up, for device
                              ; independence
MSTATE1.CLKF = CLOCK
```

Figure 4-21. Mealy State Machine Example (Continued)

Simulation

This section shows how to write a simulation section to functionally simulate your design. Functional simulation of designs is optional, but is recommended to make sure that your design works the way you intend it. PLDasm provides the following simulation capabilities:

- Event-driven simulation of combinatorial, registered, and state machine designs
- Ability to set any input, preload any register, and compare any output against an expected value.
- Ability to group signals together (form a vector) to simulate a bus
- FOR-TO and WHILE loops
- IF-THEN-ELSE control
- Generation of test vectors from simulation results for inclusion in the JEDEC file.

- Simulation history file with ability to output a subset of signals to a secondary trace file.

Simulation is specified in the Simulation section of PLDasm files. The start of the Simulation section is indicated by the keyword "SIMULATION". PLDasm automatically executes the simulation section when it is present and simulation is enabled from the PLDshell Plus **Compile/Sim** submenu.

Simulation Syntax

Figure 4-22 is a sample Simulation section of a PLDasm file. It shows most of the simulation syntax and is used as the example in the following discussion. Syntax is discussed under two headings: (1) basic commands and (2) flow control.

Basic Commands

SETF — Sets the designated inputs, states, or vectors to the specified value. This is the command that allows you to change input or feedback values. Simple simulation of combinatorial designs can be accomplished using SETF commands alone.

CLOCKF — Clocks registers high-then-low, or low-then-high. A clock initialized to a low (e.g., SETF /CLK), goes high-then-low. A clock initialized to a high (e.g., SETF CLK), goes low-then-high. Simple simulation of registered circuits can be accomplished using CLOCKF and SETF commands alone.

PRLDF — Preloads the designated registers with the specified high or low values. This is a quick way to set state machines and other registered designs to known states. Clock and control signals (OE, Clear, Preset, etc.) should be set to a known state before executing the preload command; if this is not followed, preloaded registers will immediately change to an unknown state. Figure 4-22 shows how to use this command. See "Test Vector Notes" for guidelines on using this command.

VECTOR — Assigns a group of signals to a variable name. Makes it easy to group bus signals or like signals together. This is a *superset* feature of PLDasm syntax. Precedence of output is [msb . . . lsb]. Since vector assignments are declarations, they must come first in the simulation section. Figure 4-22 shows how to use this command.

TRACE_ON and **TRACE_OFF** — By default, all inputs and outputs to a device are included in the simulation history file (.HST). If you wish to observe a subset of these signals in a secondary trace file (.TRF), use the TRACE_ON/TRACE_OFF sequence. This command sequence opens and closes a simulation trace file (.TRF) to store the specified signals. These commands can be placed anywhere in the Simulation section, but only one pair is allowed. TRACE_ON can be used with a signal and/or vector list to specify which are to be recorded in the trace file. In Figure 4-22, inputs IN1 and IN2, CLK, and

```
SIMULATION

     ; send some signals to trace file, define vector
     ; preload registers and set inputs to known state

     VECTOR NUM := [ Q3, Q2, Q1, Q0 ]
     TRACE_ON  CLK  IN1  IN2  Q0  Q1  Q2  Q3
     SETF  OE  /CLK  /IN1  /IN2
     PRLDF  /Q3  /Q2  /Q1  /Q0      ; clock and controls set
                                    ; before preload command

     ; count 9 times, disable OE between 3 and 5

     FOR j := 1 TO 9 DO
          BEGIN
               IF ( j = 3 ) THEN
                    BEGIN
                         SETF /OE
                    END
               IF ( j = 5 ) THEN
                    BEGIN
                         SETF OE
                    END
               CLOCKF CLK
          END

     ; check other combinations of inputs

     SETF  IN1  IN2
     CLOCKF CLK
     SETF /IN1  IN2
     CLOCKF CLK
     SETF  IN1 /IN2
     CLOCKF CLK

     ; close trace file

     TRACE_OFF

     ; end simulation
```

Figure 4-22. Sample Simulation Section

outputs Q0 through Q3 are to be recorded in the trace file. The OE input is not included.

CHECK — Checks the designated signals for the specified logic state and displays an error if the actual state and expected state are not the same. Figure 4-23 shows how to use this command.

Flow Control

BEGIN/END — Delimits a sequence of simulation commands; used within a conditional statement. Figure 4-23 shows examples of this syntax.

```
; FOR/TO/THEN loop including CHECK command

; loop 8 times, check for high on Q2 after 4th clock

    FOR k := 1 TO 8 DO
        BEGIN
            CLOCKF CLK
            IF ( k = 4 ) THEN
                BEGIN
                    CHECK Q2
                END
        END


; WHILE loop example

; loop until OUTA and OUTB are both low

    WHILE ( OUTA + OUTB ) DO
        BEGIN
            CLOCKF ACLK1
        END


; IF/THEN/ELSE example

; loop 16 times, check for READY = low on 5th and 6th
; clocks; check for READY = high all other times

    FOR cntr := 0 TO 15 DO
        BEGIN
            IF cntr (>=4 * <=5) THEN
                BEGIN
                    CHECK /READY
                END
            ELSE
                BEGIN
                    CHECK READY
                END
            CLOCKF CLK
        END
```

Figure 4-23. Additional Simulation Syntax Examples

FOR/TO/DO <loop> — FOR statement defines conditions under which the DO loop is executed. Figure 4-23 shows an example of this syntax.

WHILE/DO <loop> — WHILE statement defines conditions under which the DO loop is executed. Figure 4-23 shows an example of this syntax.

IF/THEN/ELSE — Provides IF/THEN/ELSE syntax for executing sequences of simulation commands. Conditions must be enclosed in parentheses. Figure 4-23 shows an example of this syntax.

ASSIGNMENTS — Vectors may be assigned any numeric constant or any Boolean expression in conjunction with a SETF command. Constants may be expressed in any of the following number systems: hexadecimal, octal, and decimal. Examples of how to specify different number systems are as follows:

hexadecimal	leading	0x	0xD51B
	or	#h	hD51B
	or	#H	#HD51B
octal	leading	0	0377
	or	#o	#o377
	or	#O	#O377
binary	leading	#b	#b1011
	or	#B	#B1011
decimal	no leading sequence		76

For example,

```
SETF VECTOR := 0xff     ; hexadecimal assignment
SETF VECTOR := a * b    ; decimal assignment
```

An example of how to use this feature is shown below:

```
;set all address of decode
VECTOR ADDR := [ a8, a7, a6, a5, a4, a3, a2, a1 ]

FOR COUNT := 0 TO 256 DO
     BEGIN
          SETF ADDR := COUNT
     END
```

EXPRESSIONS — WHILE/DO and IF/THEN/ELSE allow any combination of Boolean expression or conditional for the condition. For example,

```
IF ((A * B) :+: (C + D) THEN
```

CONDITIONALS — The following conditional operations are supported:

=	Equals
>	Greater than
<	Less than
>=	Greater than or equal to
<=	Less than or equal to
/=	Does not equal

Chapter 5 – Compiling and Simulating

This section describes design methodology, including device-independent and device-specific designs, and shows how to use the compilation and simulation features of PLDshell Plus/PLDasm.

Design Methodology

There are two basic approaches to PLD design: Device-Independent Design, and Device-Specific Design.

With Device-Independent Design, a state diagram or architectural specification is first translated into a PLDasm file without a target device in mind. This allows a designer to test a design concept by building the logic and simulation section together in an iterative loop. The design can be simulated and reworked until it is functionally correct. Once the design is working, the full compilation process can be executed to fit the design into a target device and generate a JEDEC file. Simulation can still be run during this stage to test any changes made to help fit the design into a device.

The major benefit of Device-Independent Design is that it allows a designer to concentrate more on the conceptual aspect of the design first, then on the concrete implementation. The tradeoff for this approach is that some designs can function correctly, but may not be able to fit into a specific device architecture without some changes.

With Device-Specific Design, the target device is in mind from the start. Creation of the design and fitting of the design occur together. Since a designer will already be familiar with the architecture that has been chosen, any unique features of that architecture can be comprehended as the design is developed. Fitting occurs during each pass through the compiler.

The major benefit of Device-Specific Design is that the designer knows that if the design compiles, it has already been fitted. The tradeoff for this approach is that the full fitting process must be run on each pass through the compiler.

Each approach is discussed in greater detail in the following sections.

Device-Independent Design

With Device-Independent design, the logic and simulation sections for a design are built up together without a target device in mind. During processing, the PLDasm file is parsed and the design simulated, but the design file is not fitted. The minimizer is also run if it has been selected. The simulation results are viewed to verify that the design is functioning as intended. The design can be changed until everything is working as it should. This phase of the design process is implemented via the "Simulate Only" option in the **Compile/Sim** menu.

At this point, the designer attempts fitting the design into a device using the "Compile Only" or "Compile Then Simulate" options. The "Compile Only" option runs the fitter but does not re-simulate the design. The "Compile Then Simulate" option minimizes, fits, then re-simulates the design. If the design does not fit on the first attempt, the designer enters a second iterative loop. During this phase the design is altered, not to implement new functions but to shift existing functions to better fit device resources.

Sample Design

With Device-Independent design, the designer will often begin with a state diagram of the design. An example of this is the state diagram shown in Figure 5-1. This is a 4-bit up/down counter featuring Reset and Set inputs. The design is implemented as a Moore State Machine.

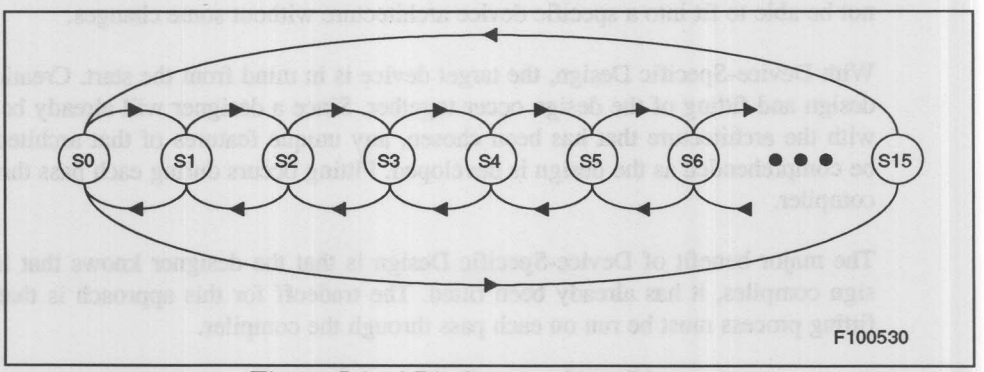


Figure 5-1. 4-Bit Counter State Diagram

Figure 5-2 is an example PLDasm listing of the design. Note that in the declaration section the CHIP line includes the reserved device name "INTEL_ARCH".

Once the design has been implemented in a PLDasm source file, the designer would process the file through the parser, minimizer (optional), and simulator. Note the simulation section of the design. For the purposes of this discussion, only the first part of the simulation section is shown. (The file called UPDOWN.PDS in your installation directory contains the complete simulation section.)

```
Title       4-Bit Up/Down Counter
Pattern     none
Revision    1.0
Author      John Doe
Company     Intel
Date        07-19-91

CHIP    updown      INTEL_ARCH

PIN     clock       ; clock input
PIN     up          ; count up
PIN     down        ; count down
PIN     reset       ; reset flip-flops
PIN     set         ; set flip-flops
PIN     q0          ; lsb
PIN     q1          ;  .
PIN     q2          ;  .
PIN     q3          ; msb

STATE  MOORE_MACHINE
     s0 = /q3 * /q2 * /q1 * /q0          s1 = /q3 * /q2 * /q1 *  q0
     s2 = /q3 * /q2 *  q1 * /q0          s3 = /q3 * /q2 *  q1 *  q0
     s4 = /q3 *  q2 * /q1 * /q0          s5 = /q3 *  q2 * /q1 *  q0
     s6 = /q3 *  q2 *  q1 * /q0          s7 = /q3 *  q2 *  q1 *  q0
     s8 =  q3 * /q2 * /q1 * /q0          s9 =  q3 * /q2 * /q1 *  q0
     sa =  q3 * /q2 *  q1 * /q0          sb =  q3 * /q2 *  q1 *  q0
     sc =  q3 *  q2 * /q1 * /q0          sd =  q3 *  q2 * /q1 *  q0
     se =  q3 *  q2 *  q1 * /q0          sf =  q3 *  q2 *  q1 *  q0
     s0:= upby1 -> s1 + dnby1 -> sf
     s1:= upby1 -> s2 + dnby1 -> s0
     s2:= upby1 -> s3 + dnby1 -> s1
     s3:= upby1 -> s4 + dnby1 -> s2
     s4:= upby1 -> s5 + dnby1 -> s3
     s5:= upby1 -> s6 + dnby1 -> s4
     s6:= upby1 -> s7 + dnby1 -> s5
     s7:= upby1 -> s8 + dnby1 -> s6
     s8:= upby1 -> s9 + dnby1 -> s7
     s9:= upby1 -> sa + dnby1 -> s8
     sa:= upby1 -> sb + dnby1 -> s9
     sb:= upby1 -> sc + dnby1 -> sa
     sc:= upby1 -> sd + dnby1 -> sb
     sd:= upby1 -> se + dnby1 -> sc
     se:= upby1 -> sf + dnby1 -> sd
     sf:= upby1 -> s0 + dnby1 -> se
```

Figure 5-2. Sample Device-Independent Design

```
CONDITIONS
    upby1 = up * /down      dnby1 = /up * down

EQUATIONS
    q0.clkf = clock     q1.clkf = clock
    q2 clkf = clock     q3.clkf = clock
    q0.rstf = reset     q1.rstf = reset
    q2.rstf = reset     q3.rstf = reset
    q0.setf = set       q1.setf = set
    q2.setf = set       q3.setf = set
    q0.trst = vcc       q1.trst = vcc
    q2.trst = vcc       q3.trst = vcc

SIMULATION
    VECTOR count := [q3,q2,q1,q0]
    SETF up /down /reset /clock
    PRLDF /q0 /q1 /q2 /q3
    SETF clock
    SETF reset /clock
    SETF /reset set
    SETF /reset /set
    CLOCKF clock
```

Figure 5-2. Sample Device-Independent Design

Figure 5-3 shows the waveforms produced by this simulation. Note that the outputs (Q0-Q3) go high immediately after SET goes high, thus implementing an asynchronous Set. Reset is also asynchronous, clearing the outputs to a logic low immediately after being asserted.

NOTE

During device-independent design, all Set and Reset signals are asynchronous. Depending on which device is eventually selected, the Set signal may be implemented synchronously. Differences between device-independent and device-dependent design are described in "Device-Independent Design Notes."

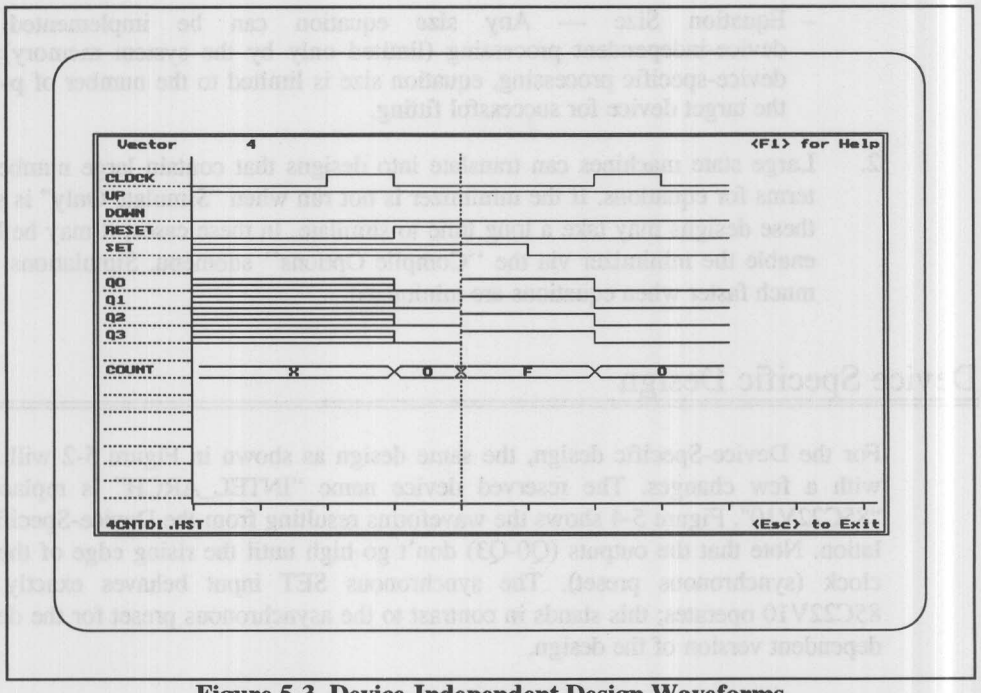


Figure 5-3. Device-Independent Design Waveforms

Device-Independent Design Notes

The following notes are provided to help you perform device-independent design:

1. Device-independent design allows you to design with no device resource limitations. This means that you can implement options that are not supported on all devices. Some redesign for fitting may eventually be needed when you start with device-independent design. Areas to be aware of include:

   - Set and Reset Signals — These are asynchronous during device-independent processing; only those actually supported by the device are valid during device-specific processing.

   - Register Types — Any register type (D, T, JK, SR) is valid during device-independent processing; only those actually supported by the device are valid during device-specific processing.

   - Input Types — Any input type (direct, latched, registered) is valid during device-independent processing; only those actually supported by the device are valid during device-specific processing.

   - Register Preloads — True register preloads are supported in the Simulation section during device-independent processing; these are converted to standard output vectors for devices that do not support preloads during device-specific processing.

- Equation Size — Any size equation can be implemented during device-independent processing (limited only by the system memory); during device-specific processing, equation size is limited to the number of p-terms in the target device for successful fitting.

2. Large state machines can translate into designs that contain large numbers of p-terms for equations. If the minimizer is not run when "Simulate Only" is selected, these designs may take a long time to simulate. In these cases, it may be better to enable the minimizer via the ''Compile Options'' submenu. Simulations will run much faster when equations are minimized.

Device Specific Design

For the Device-Specific design, the same design as shown in Figure 5-2 will be used with a few changes. The reserved device name "INTEL_ARCH" is replaced with "85C22V10". Figure 5-4 shows the waveforms resulting from the Device-Specific simulation. Note that the outputs (Q0-Q3) don't go high until the rising edge of the second clock (synchronous preset). The synchronous SET input behaves exactly as the 85C22V10 operates; this stands in contrast to the asynchronous preset for the device-independent version of the design.

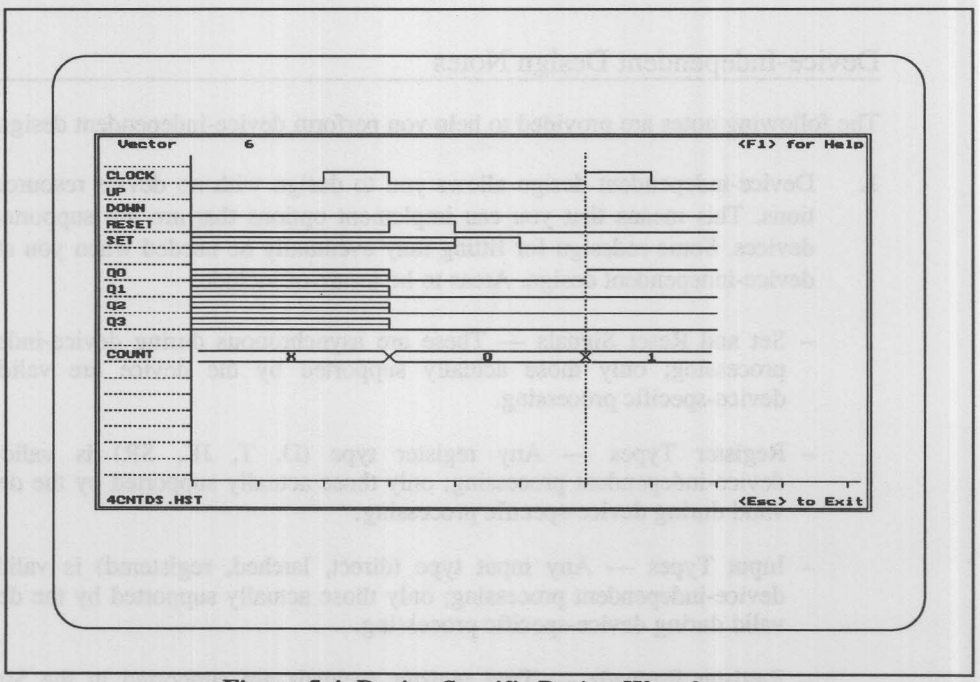


Figure 5-4. Device-Specific Design Waveforms

Design-Specific Design Notes

When performing device-specific design, the compiler and simulator operate with full knowledge of the target device characteristics. The following notes apply to compilation when the target device is selected:

1. PAL/GAL Designs — Different PLCC pinouts for devices have been adopted by some PAL/GAL manufacturers. PLDasm is only able to fit designs that use the standard pinout supported by Intel's PLCC packages. You may need to change the pin numbers in your source file when compiling designs that use PAL/GAL PLCC device names.

2. PAL/GAL Designs — PLDasm does not recognize all variations of PLCC package designators for all PAL/GAL manufacturers. To ensure recognition of PLCC names in PLDasm source files, use an "NL" designator for 20-pin packages and an "FN" designator for 28-pin packages after the base part name. See Table 7-1 for all supported device names.

3. PAL/GAL Designs — In some cases, PLDasm performs modifications to make sure that the final design performs in the same way as the original design. OE inversion is performed *automatically*; you do not need to make this change yourself. When modifications like this occur, a message is displayed on the screen and a comment is placed in the PLDasm source file.

   For example, registered PALs/GALs contain a dedicated active-low OE pin. Intel µPLDs, however, use a p-term to control the OE, even for registered designs. In this case, the software generates the appropriate p-term for an active-low OE signal. If OE is always enabled (tied to GND) in registered PAL/GAL designs, it will be always enabled in the Intel µPLD design (tied to VCC).

4. PAL/GAL Designs — PLDasm fits PAL/GAL designs into Intel µPLDs. Since µPLD architectures are supersets of PAL/GAL architectures, some designs originally targeted for Intel parts and using Intel part names will fit into Intel devices, but will not fit into PAL/GAL devices. If backward compatibility with PAL/GAL devices is a design requirement, source files should be compiled using only PAL/GAL device names.

5. All Designs — Most Intel µPLDs contain one or more Turbo Bits that optimize device performance for speed or power savings. The default state of the Turbo Bits during compilation is On, which optimizes device performance for speed. If you wish to optimize a device for power savings, you must specify TURBO=OFF in the PLDasm source file. (The erased state of Intel µPLDs is Turbo Off.)

6. All Designs — Most µPLDs have a Security Bit (sometimes called Verify Protect). The default state of the Security Bits during compilation is off, which allows programmed devices to be read. When this bit is set On, the contents of the device cannot be read. This provides a high degree of design security. To set this bit in the JEDEC file, you must specify "SECURITY = ON" in the PLDasm source file. (The erased state of Intel µPLDs is Security Bit Off.)

Using the PLDasm Compiler Options

The compiler options allow compilation for a variety of design considerations. Some combinations of compiler options are not legal. Table 5-1 lists the legal combinations. The following paragraphs explain when to use the different options.

Table 5-1. Legal Compiler Option Combinations

Expand Equations	Minimize (Espresso)	Automatic Inversion	Description
No	No	No	Do not change equations
Yes	No	No	Expand to SOP Only
Yes	Yes	No	Expand, Minimize, No Automatic Inversion
Yes	Yes	Yes	Expand, Minimize, Automatic Inversion (Default options)
Yes	No	Yes	Expand, no Minimization, Automatic Inversion
All other combinations are ILLEGAL			

Expand Equations

Normally set ON (Yes) — Expands equations to SOP (Sum-of-Products) form to allow minimization and automatic inversion. All designs must be in SOP form for minimization and fitting. In cases where equations are already in SOP form and you may not want the compiler to make any changes, set this option to NO.

Do Not Use — When this option is set to NO, designs are passed directly to the Fitter. Since the minimization and automatic inversion algorithms require SOP form, the combination of Expand Equations = NO with Minimize = YES and/or Automatic Inversion = YES is illegal.

Minimize (Espresso)

Normally set ON (Yes) — Minimizes SOP equations to least number of p-terms. The minimizer uses the ESPRESSO-IImv logic reduction algorithm, which reduces equations to the least number of p-terms almost all the time. Minimization assumes all equations are in sum-of-products form.

Do Not Use — when a design is a known fit and no changes should be made by the compiler.

Automatic Inversion

Normally set ON (Yes) — Automatically inverts equations using DeMorgan's equations if the inverted form will use fewer p-terms.

Do Not Use — When automatic inversion is not selected and the minimizer is run, DeMorgan's inversion rules can still be applied, but only at your discretion. The minimizer determines if the inverted form requires less p-terms than the original form. If so, the minimizer requests permission to use the inverted form. You can respond "Y" or "N".

NOTES

The 85C220-7 and 85C224-7 have a t_{PD} of 7.5 ns when the invert control is on and a t_{PD} of 8.5 ns when the invert control is off. To obtain faster timing for critical outputs, you must: (1) write critical equations with the invert symbol (/) on the left-hand side of the equal (=) symbol, (2) compile the design with automatic inversion = NO, and (3), if the minimizer is run, answer "N" if the minimizer requests permission to invert a critical output. Figure 5-5 shows a sample design containing two 7.5 ns outputs (fastout1 and fastout2) and one 8.5 ns output (normout1).

The 85C22V10 and iPLD22V10 have a programmable invert option after the register. Inverted forms of equations may cause designs to power-up in unexpected states. Good design practice is to include a means of reseting designs to a known state rather than depending on the power-up state.

Error File

Normally set ON (Yes) — Creates an error file (.ERR) each time the compiler or simulator is run. The Error file contains all the messages displayed during the compilation/simulation process, including information and warning messages. This file can be viewed under the **View** menu. The on-line help for each error message can be accessed.

Do Not Use — if disk space is limited

```
Title           Example of design with critical tpd
Pattern         pds
Revision
Author          E.E. Designer
Company         Intel
Date            9-1-91

CHIP U1 85C224

PIN  1    clk
PIN  2    inp1
PIN  3    inp2
PIN  4    inp3
PIN  5    inp4
PIN  6    inp5
PIN  13   oe
PIN  15   fastout1       ; this output needs least possible delay
PIN  16   fastout2       ; this output needs least possible delay
PIN  19   normout1

EQUATIONS

/fastout1 = inp1 * fastout2    ; *** ACTIVE LOW guarantees least tpd
          + inp1 * inp4
          + /inp2 * fastout2
          + /inp2 * inp4
/fastout2 = inp2               ; *** ACTIVE LOW guarantees least tpd
normout1 = fastout1 * inp4
fastout1.trst = oe
```

Figure 5-5. 85C224-7 t_{PD} Example Design

Report File

Normally set ON (Yes) — Creates the utilization report file (.RPT) each time the compiler is run. The Report file shows the final pinout of the design and shows how resources are assigned in the final, fitted design. Appendix B discusses the different sections of the Report files.

Do Not Use — if disk space is limited

Fitter Options

There are three Fitter options:

Use Pin Assignments, Abort on no Fit

This option is used when the final pinout for a design is already fixed (e.g., the target board is already layed out). The Fitter will try to fit all resources to the pins declared in the source file. If any pin does not fit, the process aborts with an error message. This is the default fitting option.

Use Pin Assignments, But Reassign if Needed

This option gives first preference to pin assignments provided by the designer. If a design will not fit, the Fitter ignores the pin assignments for that design and will use its auto-fit algorithms.

Ignore All Pin Assignments

This option requires the Fitter to use its automatic fitting algorithms to fit a design. It can eliminate the need for the designer to know all the details of the device pinout. All pin assignments in the source file are ignored. The Fitter will abort if all attempts have failed. Note that the automatic fitting algorithms make some assumptions about the information provided for each output. These assumptions are described in the "Automatic Pin Assignment" section in Chapter 4.

Using the Simulation Options

The Simulation options allow the designer to show asynchronous events during stabilization of outputs during each simulation step and set the maximum number of such events. This is very useful in identifying race/glitch conditions in combinatorial or asynchronously clocked designs and in tracking design problems in circuits that make extensive use of feedbacks.

Show Asynchronous Events

Normally set to OFF (No) — Normally not used unless a design problem is suspected.

Use — when you want to locate and analyze problems in the design (for example, to analyze why outputs do not appear to function correctly). Can also be used to observe the detailed operation of asynchronously clocked state machines.

Threshold

Normally set to 32 — The threshold range is 1 to 32,767. The default value is normally adequate to analyze a design problem. Adjust the threshold value as necessary to display the number of events to provide enough detail to analyze the problem. For example, a design with numerous asynchronous events should use a larger number.

Viewing Simulation Output Files

The PLDasm simulator generates two output files: (1) a history file (.HST) containing simulation results for all signals for the entire simulation, and (2) an optional trace file (.TRF) containing simulation results for the specified signals for a defined part of the simulation.

The .HST file can be viewed as state table (1s and 0s) or as waveform output under the View menu. Waveforms are viewed in graphics mode and printed in text mode. Text mode printing uses standard ASCII characters or the extended IBM/DOS graphic character set.

Figure 5-6 shows state table output for the simulation results generated by the Simulation section in Figure 4-17. This is the whole history file as it appears when output to a printer.

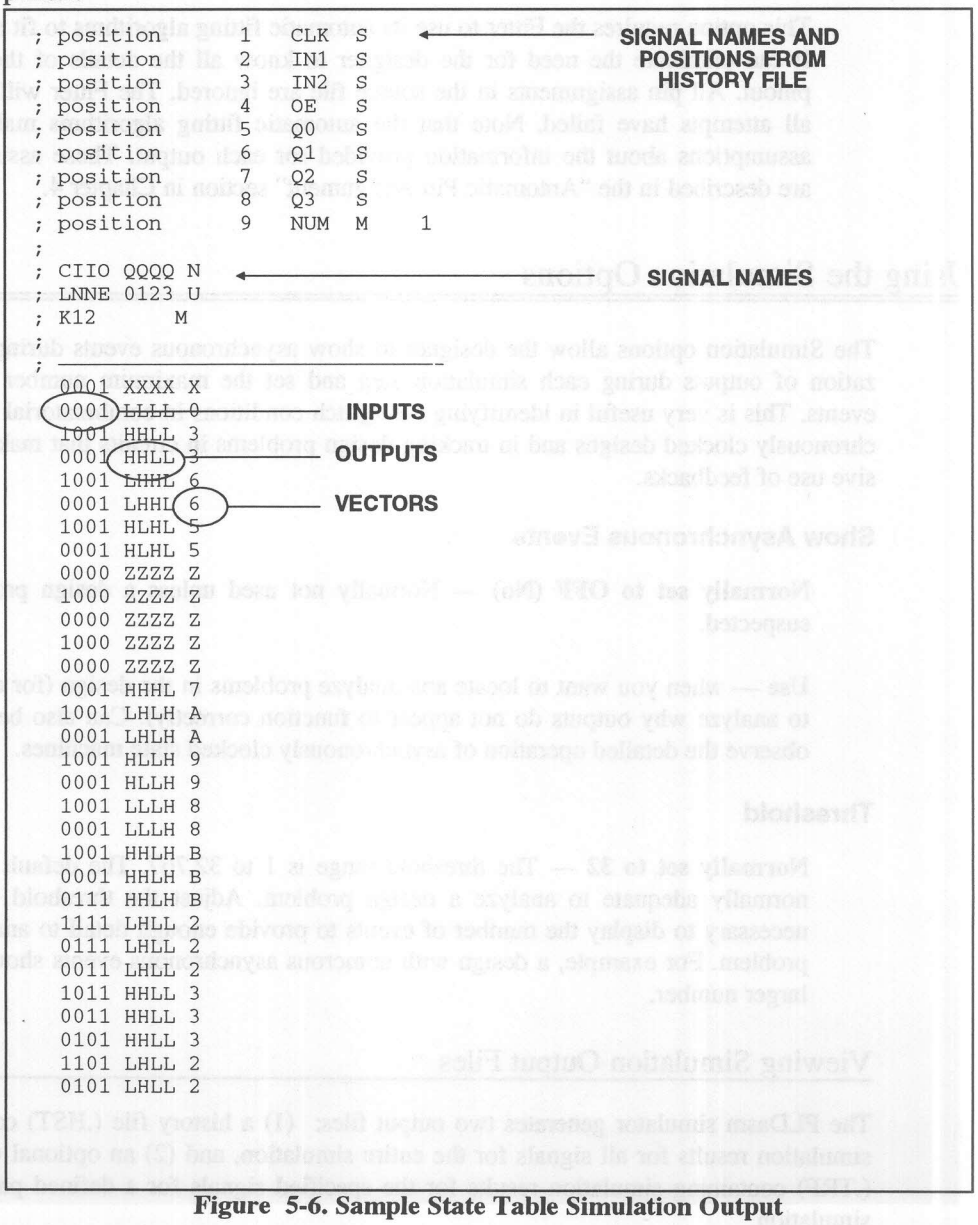


Figure 5-6. Sample State Table Simulation Output

Many programmers have the ability to use test vectors to perform a functional test on PLDs after programming. Input and output values from the .HST file are also used to generate these test vectors for the JEDEC file. Notes on test vectors are provided in "Test Vector Notes."

The .TRF file can be used to isolate the critical signals you need to check during simulation. This file is output in state table (1s and 0s) format only.

Simulation Notes

The following notes describe some behaviors of the simulator that may be important for certain types of designs:

1. If the data to a register changes state at the same time as a valid clock edge occurs, one of the following behaviors will occur:

   a) For synchronously clocked registers (i.e., registers clocked by the dedicated clock pin), the data presented to the register input will *not* be clocked through the register during the current clock cycle. This behavior emulates the synchronous clocking behavior of PLDs, which require that data meet a setup time to the clock edge.

   b) For asynchronously clocked registers (i.e., registers clocked by a p-term from the logic array), the data presented to the register input will be clocked through the register during the current clock cycle. The behavior emulates the asynchronous clocking behavior of PLDs, which have a much shorter (or no) setup time to the clock edge.

2. Executing the SETF command on a vector without an assignment sets all bits of the vector to the specified logic level (logic 1 or logic 0). For example, the following sequence declares a vector (NUM), sets all bits of NUM to 1, then clears all bits of NUM to 0.

```
VECTOR NUM = [ Q3 Q2 Q1 Q0 ]
SETF NUM
SETF /NUM
```

   A more appropriate way is to assign the vector as follows:

```
VECTOR NUM = [ Q3 Q2 Q1 Q0 ]
SETF NUM := 0xF
SETF NUM := 0x0
```

3. A .FB extension can be used to simulate different logic states on a feedback path and output. The example below shows simulation of pin feedback on I/O pins. In the example, OE is disabled for pin OUTA (an I/O pin). The feedback is then driven via OUTA.FB:

```
SETF      /OE           ; disable OE
SETF      OUTA.FB       ; drive I/O feedback high
SETF      OE            ; driving I/O pin
```

The .FB extension does not need to be defined in the pin declarations or in any of the design sections It is automatically understood by the simulator.

4. If a valid clock edge occurs during a register preload, the register states will be indeterminate.

Test Vector Notes

The following notes provide information that can help you create test vectors for use during programming verification.

1. The 85C22V10 and iPLD22V10 support true register preloads during programming. Preloads in the simulation vector file are converted to JEDEC preload test vectors. If the programmer does not support preloads, you may encounter an error when performing a program test.

2. With the exception of the 85C22V10 and iPLD22V10, the test vectors in the JEDEC file for the preload simulation command (PRLDF) are simple output state values (SETFs), not true preload values. If you plan to use test vectors during programming for other devices, use the preload command only once at the beginning of the Simulation section to set registers to their power-up state. Use of the preload command elsewhere or with values other than the power-up value will cause warnings during the fitting process. Test vectors beginning with an illegal preload are not placed in the JEDEC file to eliminate the chance of programming errors. Note, however, that the programming test in this case can be much less thorough than expected.

3. During compilation, test vectors are automatically generated from the .HST file with the same base filename of the PLDasm file. If you do not include a simulation section in your current design file, but have a .HST file from a previous design that used the same filename, PLDasm will generate test vectors from the earlier design and include them in the JEDEC file for the new design. This will probably cause programming test failures. Deleting old .HST files when a design is complete will prevent this problem from occurring.

Chapter 6 – Using the Utility Programs

PLDshell Plus provides utility programs for processing files between different source and target file formats. This section describes how to use the Disassembly, Conversion, and Translation utilities.

Disassembly

The disassembly program processes a JEDEC file and creates a PDS source file. The JEDEC file can be for a supported non-Intel device, however, the resulting PDS source file will be for an Intel device only. Disassembly of Intel µPLD JEDECs is also supported.

The options available on the Disassemble submenu are:

Input Filename — This is the JEDEC input file with the .JED extension.

Source Device — This is the source device for which the JEDEC file was originally generated. Pressing the <SPACE> key displays a list of supported source devices. Once the source device is selected, the target Intel device is automatically selected and displayed on the right.

Package Type — Sets the package type. Pressing the <SPACE> key displays a list of package types for the device selected in the Source Device field.

Output Filename — Displays the output filename that will be created during disassembly. The default is the target device name plus the .PDS extension.

```
Disassembling...
INFO JDBMAC: 85C224 JEDEC Disassembly in Progress...
Processing Macrocell [1]
Processing Macrocell [2]
Processing Macrocell [3]
Processing Macrocell [4]
Processing Macrocell [5]
Processing Macrocell [6]
Processing Macrocell [7]
Processing Macrocell [8]

INFO JDBMAC: Disassembly Successfully Completed.

Disassembly Successfully Completed.

Press ENTER to Continue...
```

Figure 6-1. Disassembly Messages

Figure 6-1 shows the messages displayed during disassembly. Figure 6-2 shows a .PDS file created from the 4COUNT.JED file (created by compiling the 4COUNT.PDS file in your installation directory). This is a simple design using two inputs and four I/O macrocells to implement a 4-bit counter.

```
; OPTIONS TURBO=OFF

CHIP U1 85C224

PIN  1    in1       ; pin 1 is synchronous clock for any
                    ; registers.
PIN  2    in2
PIN  3    NC
PIN  4    NC
PIN  5    NC
PIN  6    NC
PIN  7    NC
PIN  8    NC
PIN  9    NC
PIN  10   NC
PIN  11   NC
PIN  12   GND
PIN  13   NC
PIN  14   NC
PIN  15   io15
PIN  16   io16
PIN  17   io17
PIN  18   io18
PIN  19   NC
PIN  20   NC
PIN  21   NC
PIN  22   NC
PIN  23   NC
PIN  24   VCC

EQUATIONS

io18 := in2 * /io15 * io18
     + in2 * /io16 * io18
     + in2 * /io17 * io18
     + in2 * io15 * io16 * io17 * /io18
io18.TRST = VCC
io17 := in2 * /io15 * io17
     + in2 * /io16 * io17
     + in2 * io15 * io16 * /io17
io17.TRST = VCC
io16 := in2 * /io15 * io16
     + in2 * io15 * /io16
io16.TRST = VCC
io15 := in2 * /io15
io15.TRST = VCC
```

Figure 6-2. Disassembled 4COUNT.JED File

Disassembly Notes

The following notes apply to JEDEC disassembly:

1. For All Designs — During generation of a PLDasm source file, the disassembly program creates signal names based on the pin to which signals are mapped, e.g., "in8" means "input pin 8", "io13" means "I/O pin 13", etc. You may wish to edit the resulting PLDasm source file to change the signal names before compiling the file.

2. For PAL/GAL Designs — Different pinouts have been adopted for PLCC devices by some PAL/GAL manufacturers. PLDshell Plus normalizes these different pinouts to the standard pinout supported by Intel's PLCC packages. This is referred by most manufacturers as an ''NL'' package for 20-pin packages and an "FN" package for 28-pin packages. Please note this possible pinout difference when disassembling designs from PAL/GAL JEDEC files.

3. For PAL/GAL Designs — In some cases, modifications are performed during disassembly to ensure that the final design performs in the same way as the original design. OE inversion is performed *automatically*; you do not need to make this change yourself. When modifications like this occur, a message is displayed on the screen and a comment is placed in the PLDasm source file.

   For example, registered PALs contain a dedicated active-low OE pin. Intel µPLDs, however, use a p-term to control the OE, even for registered designs. In this case, the software generates the appropriate p-term for a active-low OE signal. If OE is always enabled (tied to GND) in registered PAL/GAL designs, it will be always enabled in the Intel µPLD design (tied to VCC).

4. For PAL/GAL Designs — JEDEC disassembly creates .PDS files that fit into Intel µPLDs. Since µPLD architectures are supersets of PAL/GAL architectures, some designs converted to .PDS files for Intel devices will no longer compile for PAL/GAL devices using other logic compilers. If backward compatibility with PAL/GAL devices is a design requirement, .PDS files should be compiled using both PLDasm and your existing logic compiler.

5. For All Designs — Most Intel µPLDs contain one or more Turbo Bits that optimize device performance for speed or power savings. The default state of the Turbo Bits during disassembly is On, which optimizes device performance for speed. If you wish to optimize a device for power savings, you must edit the resulting PLDasm source file to specify ''TURBO=OFF''. (The erased state of Intel µPLDs is Turbo Off.)

6. For PAL/GAL Designs — JEDEC files for PALs/GALs using early versions of PALASM software do not contain a field called the fuse count field (QF field) or have the fuse count field in a non-standard position. This will cause an error message to be displayed early during the disassembly. To correct this error, open the

JEDEC file using a text editor and insert the proper QF field (see Table 6-1). The fuse field must be inserted immediately after the header record (i.e., immediately after the first "*"). The asterisk after the fuse count is required.

Table 6-1. Fuse Count (QF) Fields

Part	Fuse Field
16L8	QF2048*
16R4	QF2048*
16R6	QF2048*
16R8	QF2048*
16V8	QF2194*
20L8	QF2560*
20R4	QF2560*
20R6	QF2560*
20R8	QF2560*
20V8	QF2706*
22V10	QF5828*
22VP10	QF5838*
22V10ES	QF5892*

JEDEC Conversion

PLDshell Plus provides the capability to convert existing JEDEC files for common PALs/GALs into JEDEC files for Intel µPLDs. A PLDasm source file is generated as a part of the conversion process. (JEDEC conversion consists of disassembly followed automatically by compilation.)

The options available on the Convert JEDEC submenu are:

Input Filename — Displays a list of JEDEC files to convert. The default extension is .JED.

Source Device — Sets the source device to be used. Pressing the <SPACE> key displays a list of devices. When the source device has been selected, the target Intel device is automatically selected and displayed on the right.

Package Type — Sets the package type. Pressing the <SPACE> key displays a list of package types for the device selected in the Source Device field.

Output Filename — Displays the output filename that will be created during conversion. The default name is the target device name with a .JED extension.

Figure 6-3 shows the messages displayed during JEDEC conversion process. These messages were generated while converting the file EX20V8.JED in your installation directory. This design uses all available inputs and outputs.

```
Running Conversion..
INFO JDBMAC: 20V8 -> 85C224 Disassembly in progress
Processing Macrocell [1]
Processing Macrocell [2]
Processing Macrocell [3]
Processing Macrocell [4]
Processing Macrocell [5]
Processing Macrocell [6]
Processing Macrocell [7]
Processing Macrocell [8]
INFO JDBMAC: JEDEC Disassembly Successfully Completed.
INFO PARPDS: Parsing file: 85C224.pds
INFO PARPDS: File parsed correctly.
INFO PARPDS: Equation Expansion Complete.
INFO FIT: Design compiled into 85C224.
INFO ASM: JEDEC file Assembled.

Disassembly Successfully Completed.

Press ENTER to Continue...
```

Figure 6-3. JEDEC Conversion Messages

Conversion Notes

The following notes apply to JEDEC conversion:

1. For All Designs — During generation of a PLDasm source file, PLDshell Plus creates signal names based on the pin to which signals are mapped, e.g., "in8" means "input pin 8", "io13" means "I/O pin 13", etc. You may wish to edit the resulting PLDasm source file to change the signal names, then recompile the file.

2. For PAL/GAL Designs — Different pinouts have been adopted for PLCC devices by some PAL/GAL manufacturers. PLDshell Plus normalizes these different pinouts to the standard pinout supported by Intel's PLCC packages. Please note this possible pinout difference when converting designs from PAL/GAL JEDEC files.

3. For PAL/GAL Designs — In some cases, modifications are performed during disassembly to ensure that the final design performs in the same way as the original design. OE inversion is performed *automatically*; you do not need to make this change yourself. When modifications like this occur, a message is displayed on the screen and a comment is placed in the PLDasm source file.

For example, registered PALs/GALs contain a dedicated active-low OE pin. Intel µPLDs, however, use a p-term to control the OE, even for registered designs. In this case, the software generates the appropriate p-term for an active-low OE signal. If OE is always enabled (tied to GND) in registered PAL/GAL designs, it will be always enabled in the Intel µPLD design (tied to VCC).

4. For PAL/GAL Designs — JEDEC conversion creates .PDS and .JED files for Intel µPLDs. Since µPLD architectures are supersets of PAL/GAL architectures, some designs converted to .PDS files for Intel devices will no longer compile for PAL/GAL devices using other logic compilers. If backward compatibility with PAL/GAL devices is a design requirement, .PDS files should be compiled using both PLDasm and your existing logic compiler.

5. For All Designs — Most Intel µPLDs contain one or more Turbo Bits that optimize device performance for speed or power savings. The default state of the Turbo Bits during compilation is On, which optimizes device performance for speed. If you wish to optimize a device for power savings, you must edit the resulting PLDasm source file to specify TURBO=OFF, then recompile the source file using the **Compile/Sim** menu selection. (The erased state of Intel µPLDs is Turbo Off.)

6. For PAL/GAL Designs — JEDEC files for PALs/GALs using earlier versions of PALASM software do not contain a field called the fuse count field (QF field) or have the fuse count field in a non-standard position. This will cause an error message to be displayed early during the disassembly. To correct this error, open the JEDEC file using a text editor and insert the proper QF field (see Table 6-1). The fuse field must be inserted immediately after the header record (i.e., immediately after the first "*"). The asterisk after the fuse count is required.

7. During JEDEC conversion, the PLDasm compiler is run with Expand Equations = Yes, Minimize = No, and Automatic Inversion = No. If you want to compile with different options, recompile the intermediate .PDS file from the Compile/Sim menu.

Translation

PLDshell Plus provides the capability of translating ADF/SMF source files (originally written for iPLS II) into PDS source files that can be compiled by PLDasm. Figure 6-4 shows translation flow.

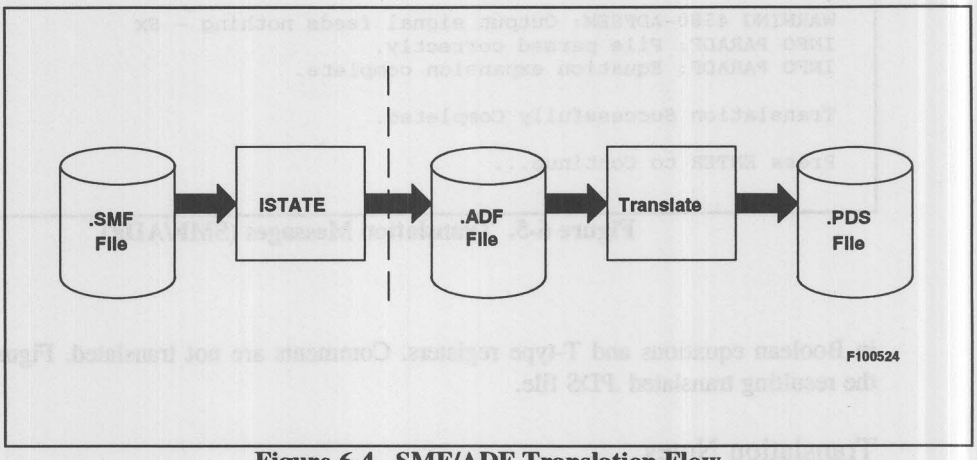


Figure 6-4. SMF/ADF Translation Flow

The options on the Translate submenu are:

Input Filename — Displays a list of ADF/SMF files to be translated. The input filename can have any legal basename, but *must* have a .ADF or .SMF extension. The default extension is .ADF; for SMF files, you can type in the .SMF extension.

Output Filename — Displays the filename selected in the Input Filename field but with a .PDS extension.

Example Translation

Figure 6-5 shows the messages displayed during translation of a .SMF file to an .ADF then to a .PDS file. Note the message lines,

```
WARNING 4380-ADFSEM: Output signal feeds nothing - Sx
```

These messages are displayed as redundant state name entries are removed during translation; they can be ignored.

Figure 6-6 shows the COUNT.SMF file, which is in your install directory. This is a 32-bit binary counter targetted for an 85C060. Figure 6-7 shows the file COUNT.ADF, which can be created by translating the .SMF file. The original state machine syntax has been converted to comments and the state machine functionally has been implemented

```
Translating...

INFO ISTATE: CONVERTING COUNT.SMF to COUNT.adf

INFO PARADF: Parsing file: COUNT.ADF
WARNING 4380-ADFSEM: Output signal feeds nothing - Sx
.
.
.
WARNING 4380-ADFSEM: Output signal feeds nothing - Sx
INFO PARADF: File parsed correctly.
INFO PARADF: Equation expansion complete.

Translation Successfully Completed.

Press ENTER to Continue...
```

Figure 6-5. Translation Messages (SMF/ADF)

in Boolean equations and T-type registers. Comments are not translated. Figure 6-8 shows the resulting translated .PDS file.

Translation Notes

The following notes apply to ADF/SMF-to-PDS translation:

1. During translation, state machines and truth tables converted to Boolean equations as they are loaded. Thus, the .PDS source file generated by the translation utility will not contain state machine and truth table syntax. The original state names and truth table names, however, are preserved in the Boolean equations.

2. Macro calls in ADF/SMF designs are expanded into their Boolean equation implementations during translation. The original macros appear in the .PDS file as comments to help in tracing the translation process. The original signal names are preserved.

3. Comments in the ADF/SMF input files are not translated.

4. If not specified in the ADF/SMF file, TURBO=ON is set in the PDS options section. This is a change from iPLS II, where the default was TURBO=OFF if the state of the Turbo Bit was not specified.

```
Name
Intel
January 13, 1988
1
a
5C060
Large Counter
OPTIONS: TURBO=ON
PART: 5C060
INPUTS: CLK
OUTPUTS: SM0, SM1, SM2, SM3, SM4
NETWORK:
     CLK = INP(CLK)
MACHINE: COUNTER
  CLOCK: CLK
  STATES:    [SM4  SM3  SM2  SM1  SM0]
       S0    [ 0    0    0    0    0 ]
       S1    [ 0    0    0    0    1 ]
       S2    [ 0    0    0    1    0 ]
       S3    [ 0    0    0    1    1 ]
       S4    [ 0    0    1    0    0 ]
       S5    [ 0    0    1    0    1 ]
       S6    [ 0    0    1    1    0 ]
       S7    [ 0    0    1    1    1 ]
       S8    [ 0    1    0    0    0 ]
       S9    [ 0    1    0    0    1 ]
       S10   [ 0    1    0    1    0 ]
       S11   [ 0    1    0    1    1 ]
       S12   [ 0    1    1    0    0 ]
       S13   [ 0    1    1    0    1 ]
       S14   [ 0    1    1    1    0 ]
       S15   [ 0    1    1    1    1 ]
       S16   [ 1    0    0    0    0 ]
       S17   [ 1    0    0    0    1 ]
       S18   [ 1    0    0    1    0 ]
       S19   [ 1    0    0    1    1 ]
       S20   [ 1    0    1    0    0 ]
       S21   [ 1    0    1    0    1 ]
       S22   [ 1    0    1    1    0 ]
       S23   [ 1    0    1    1    1 ]
       S24   [ 1    1    0    0    0 ]
       S25   [ 1    1    0    0    1 ]
       S26   [ 1    1    0    1    0 ]
       S27   [ 1    1    0    1    1 ]
       S28   [ 1    1    1    0    0 ]
       S29   [ 1    1    1    0    1 ]
       S30   [ 1    1    1    1    0 ]
       S31   [ 1    1    1    1    1 ]
```

Figure 6-6. COUNT.SMF Sample File Listing

```
S0: S1
S1: S2
S2: S3
S3: S4
S4: S5
S5: S6
S6: S7
S7: S8
S8: S9
S9: S10
S10: S11
S11: S12
S12: S13
S13: S14
S14: S15
S15: S16
S16: S17
S17: S18
S18: S19
S19: S20
S20: S21
S21: S22
S22: S23
S23: S24
S24: S25
S25: S26
S26: S27
S27: S28
S28: S29
S29: S30
S30: S31
S31: S0
END$
```

Figure 6-5. COUNT.SMF Sample File Listing (Continued)

```
Name
Intel
January 13, 1988
1
a
5C060
Large Counter
iSTATE Release XX, YY 2.22
OPTIONS: TURBO=ON
PART: 5C060
INPUTS:
CLK
OUTPUTS:
SM0, SM1, SM2, SM3, SM4
NETWORK:
CLK = INP(CLK)

%
I/O's for State Machine "COUNTER"
%
SM4, SM4 = TOTF(SM4.t, CLK, GND, GND, VCC)
SM3, SM3 = TOTF(SM3.t, CLK, GND, GND, VCC)
SM2, SM2 = TOTF(SM2.t, CLK, GND, GND, VCC)
SM1, SM1 = TOTF(SM1.t, CLK, GND, GND, VCC)
SM0, SM0 = TOTF(SM0.t, CLK, GND, GND, VCC)
EQUATIONS:
%
Boolean Equations for State Machine "COUNTER"
%
%
Current State Equations for "COUNTER"
%
S0 = SM4'*SM3'*SM2'*SM1'*SM0';
S1 = SM4'*SM3'*SM2'*SM1'*SM0;
S2 = SM4'*SM3'*SM2'*SM1*SM0';
S3 = SM4'*SM3'*SM2'*SM1*SM0;
S4 = SM4'*SM3'*SM2*SM1'*SM0';
S5 = SM4'*SM3'*SM2*SM1'*SM0;
S6 = SM4'*SM3'*SM2*SM1*SM0';
S7 = SM4'*SM3'*SM2*SM1*SM0;
S8 = SM4'*SM3*SM2'*SM1'*SM0';
S9 = SM4'*SM3*SM2'*SM1'*SM0;
S10 = SM4'*SM3*SM2'*SM1*SM0';
S11 = SM4'*SM3*SM2'*SM1*SM0;
S12 = SM4'*SM3*SM2*SM1'*SM0';
S13 = SM4'*SM3*SM2*SM1'*SM0;
S14 = SM4'*SM3*SM2*SM1*SM0';
S15 = SM4'*SM3*SM2*SM1*SM0;
S16 = SM4*SM3'*SM2'*SM1'*SM0';
S17 = SM4*SM3'*SM2'*SM1'*SM0;
S18 = SM4*SM3'*SM2'*SM1*SM0';
S19 = SM4*SM3'*SM2'*SM1*SM0;
S20 = SM4*SM3'*SM2*SM1'*SM0';
S21 = SM4*SM3'*SM2*SM1'*SM0;
S22 = SM4*SM3'*SM2*SM1*SM0';
S23 = SM4*SM3'*SM2*SM1*SM0;
S24 = SM4*SM3*SM2'*SM1'*SM0';
```

Figure 6-6. COUNT.ADF Input File Listing

```
S25 = SM4*SM3*SM2'*SM1'*SM0;
S26 = SM4*SM3*SM2'*SM1*SM0';
S27 = SM4*SM3*SM2'*SM1*SM0;
S28 = SM4*SM3*SM2*SM1'*SM0';
S29 = SM4*SM3*SM2*SM1'*SM0;
S30 = SM4*SM3*SM2*SM1*SM0';
S31 = SM4*SM3*SM2*SM1*SM0;
%
SV Defining Equations for State Machine "COUNTER"
%
SM4.t = S15 * S16.n
      + S31 * S0.n;
SM3.t = S7 * S8.n
      + S15 * S16.n
      + S23 * S24.n
      + S31 * S0.n;
SM2.t = S3 * S4.n
      + S7 * S8.n
      + S11 * S12.n
      + S15 * S16.n
      + S19 * S20.n
      + S23 * S24.n
      + S27 * S28.n
      + S31 * S0.n;
SM1.t = S1 * S2.n
      + S3 * S4.n
      + S5 * S6.n
      + S7 * S8.n
      + S9 * S10.n
      + S11 * S12.n
      + S13 * S14.n
      + S15 * S16.n
      + S17 * S18.n
      + S19 * S20.n
      + S21 * S22.n
      + S23 * S24.n
      + S25 * S26.n
      + S27 * S28.n
      + S29 * S30.n
      + S31 * S0.n;
SM0.t = VCC;
%
Next State Equations for State Machine "COUNTER"
%
S2.n = (S1);
S4.n = (S3);
S6.n = (S5);
S8.n = (S7);
S10.n = (S9);
S12.n = (S11);
S14.n = (S13);
S16.n = (S15);
S18.n = (S17);
S20.n = (S19);
S22.n = (S21);
S24.n = (S23);
S26.n = (S25);
```

Figure 6-6. COUNT.ADF Input File Listing (Continued)

```
S28.n = (S27);
S30.n = (S29);
S0.n = (S31);
END$
```

Figure 6-6. COUNT.ADF Input File Listing (Continued)

```
Name
Intel
January 13, 1988
1
a
5C060
Large Counter
iSTATE Release XXX, YYY

OPTIONS
    TURBO = ON
CHIP  5C060 5C060

PIN         CLK
PIN         SM0
PIN         SM1
PIN         SM2
PIN         SM3
PIN         SM4

EQUATIONS
SM4.T := /SM4 * SM3 * SM2 * SM1 * SM0
       + SM4 * SM3 * SM2 * SM1 * SM0
SM4.CLKF = CLK
SM4.RSTF = GND
SM4.SETF = GND
SM4.TRST = VCC
SM3.T := /SM4 * /SM3 * SM2 * SM1 * SM0
       + /SM4 * SM3 * SM2 * SM1 * SM0
       + SM4 * /SM3 * SM2 * SM1 * SM0
       + SM4 * SM3 * SM2 * SM1 * SM0
SM3.CLKF = CLK
SM3.RSTF = GND
SM3.SETF = GND
SM3.TRST = VCC
SM2.T := /SM4 * /SM3 * /SM2 * SM1 * SM0
       + /SM4 * /SM3 * SM2 * SM1 * SM0
       + /SM4 * SM3 * /SM2 * SM1 * SM0
       + /SM4 * SM3 * SM2 * SM1 * SM0
       + SM4 * /SM3 * /SM2 * SM1 * SM0
       + SM4 * /SM3 * SM2 * SM1 * SM0
       + SM4 * SM3 * /SM2 * SM1 * SM0
       + SM4 * SM3 * SM2 * SM1 * SM0
SM2.CLKF = CLK
SM2.RSTF = GND
```

Figure 6-7. COUNT.PDS Output File Listing

```
SM2.SETF = GND
SM2.TRST = VCC
SM1.T := /SM4 * /SM3 * /SM2 * /SM1 * SM0
       + /SM4 * /SM3 * /SM2 * SM1 * SM0
       + /SM4 * /SM3 * SM2 * /SM1 * SM0
       + /SM4 * /SM3 * SM2 * SM1 * SM0
       + /SM4 * SM3 * /SM2 * /SM1 * SM0
       + /SM4 * SM3 * /SM2 * SM1 * SM0
       + /SM4 * SM3 * SM2 * /SM1 * SM0
       + /SM4 * SM3 * SM2 * SM1 * SM0
       + SM4 * /SM3 * /SM2 * /SM1 * SM0
       + SM4 * /SM3 * /SM2 * SM1 * SM0
       + SM4 * /SM3 * SM2 * /SM1 * SM0
       + SM4 * /SM3 * SM2 * SM1 * SM0
       + SM4 * SM3 * /SM2 * /SM1 * SM0
       + SM4 * SM3 * /SM2 * SM1 * SM0
       + SM4 * SM3 * SM2 * /SM1 * SM0
       + SM4 * SM3 * SM2 * SM1 * SM0
SM1.CLKF = CLK
SM1.RSTF = GND
SM1.SETF = GND
SM1.TRST = VCC
SM0.T := VCC
SM0.CLKF = CLK
SM0.RSTF = GND
SM0.SETF = GND
SM0.TRST = VCC
```

Figure 6-6. COUNT.PDS Output File Listing (Continued)

Chapter 7 – Device Descriptions

This Chapter describes Intel µPLDs supported by PLDasm. The global architecture and macrocell architecture for each of these devices is discussed, together with possible device configurations. Detailed information for all Intel µPLDs is provided in the *Programmable Logic Handbook* and related publications. Tables at the start of this chapter summarize device features and list part and package names.

The information describes Intel's high-speed iPLDxxx and 85Cxxx families, the 5ACxxx Advanced Architecture family, and the 5Cxxx Standard Architecture family of µPLDs.

Device Names/Feature Summary

Table 7-1 lists the supported Intel µPLD and PAL designations and package types. Table 7-2 lists the features of Intel µPLDs.

Table 7-1. Supported PLDs, Packages, and Device Names

Intel μPLD	Package	Device Name	Compiled To For JEDEC
Device-Independent (None)	None	INTEL_ARCH	"Simulate Only"; No JEDEC Created
iPLD610	DIP	iPLD610	iPLD610
	PLCC	iPLD610N	iPLD610N
iPLD910	DIP	iPLD910	iPLD910
	PLCC	iPLD910N	iPLD910N
iPLD22V10	DIP	iPLD22V10	iPLD22V10
	PLCC	iPLD22V10N	iPLD22V10N
85C220	DIP	85C220	85C220
	PLCC	N85C220	N85C220
85C224	DIP	85C224	85C224
	PLCC	N85C224	N85C224
85C22V10	DIP	85C22V10	85C22V10
	PLCC	N85C22V10	N85C22V10
85C060	DIP	85C060	85C060
	PLCC	N85C060	N85C060
85C090	DIP	85C090	85C090
	PLCC	N85C090	N85C090
85C508	DIP	85C508	85C508
	PLCC	N85C508	N85C508
5AC312	DIP	5AC312	5AC312
	PLCC	N5AC312	N5AC312
5AC324	DIP	5AC324	5AC324
	PLCC	N5AC324	N5AC324
5C031	DIP	5C031	5C031
5C032	DIP	5C032	5C032
5C060	DIP	5C060	5C060
	PLCC	N5C060	N5C060
5C090	DIP	5C090	5C090
	PLCC	N5C090	N5C090
5C180	PLCC	N5C180	N5C180

Note: The iPLD16V8XP and iPLD20V8XP are not directly supported by PLDshell Plus. These parts use a standard 16V8 or 20V8 JEDEC file and cross-programming algorithms on your Data I/O or other third-party programmer.

Table 7-1. Supported PLDs, Packages, and Device Names

Other PLDs	Package	Device Name	Compiled to For JEDEC
16L8	DIP	16L8	85C220
	PLCC	16L8NL	N85C220
16R4	DIP	16R4	85C220
	PLCC	16R4NL	N85C220
16R6	DIP	16R6	85C220
	PLCC	16R6NL	N85C220
16R8	DIP	16R8	85C220
	PLCC	16R8NL	N85C220
16V8	DIP	16V8	85C220
	PLCC	16V8NL	N85C220
20L8	DIP	20L8	85C224
	PLCC	20L8FN	N85C224
20R4	DIP	20R4	85C224
	PLCC	20R4FN	N85C224
20R6	DIP	20R6	85C224
	PLCC	20R6FN	N85C224
20R8	DIP	20R8	85C224
	PLCC	20R8FN	N85C224
20V8	DIP	20V8	85C224
	PLCC	20V8FN	N85C224
22V10	DIP	22V10	iPLD22V10
	PLCC	22V10FN	iPLD22V10N
22VP10	DIP	22V10	85C22V10
	PLCC	22V10FN	N85C22V10

Table 7-2. Intel PLD Feature Comparison

	5C031	85C220 5C032	85C224	IPLD22V10 85C22V10	iPLD610 85C060 5C060	iPLD910 85C090 5C090	5C180	5AC312	5AC324
INPUTS									
Dedicated	10	10	14	12	4	12	12	10	12
Maximum	18	18	22	22	20	36	60	22	36
Input Latches/ Registers								Y	Y
I/O									
Number	8	8	8	10	16	24	48	12	24
Tri-State	Y	Y	Y	Y	Y	Y	Y	Y	Y
Programmable Polarity	Y	Y	Y	Y	Y	Y	Y	Y	Y
Dual-Feedback							Y(1)	Y	Y
MACROCELLS	8	8	8	10	16	24	48	12	24
REGISTERS									
Number	8	8	8	10	16	24	48	12	24
Types	D	D	D	D	D/T/RS/ JK	D/T/RS/ JK	D/T/RS/ JK	D/T/RS/ JK	D/T/RS/ JK
By-Pass	Y	Y	Y	Y	Y	Y	Y	Y	Y
Reset P-term	Y(2)			Y(2)	Y	Y	Y	Y	Y
Preset P-term	Y(2)			Y(2)				Y	Y
PRODUCT TERMS									
Number	74	72	72	132	160	240	480	200	394
Allocation								Y	Y
LOCAL/GLOBAL BUSES							Y		
CLOCKS	1	1	1	1(4)	2	2	4	2	2
Asynchronous Clocking					Y	Y	Y	Y	Y
SECURITY BIT	Y	Y	Y	Y	Y	Y	Y	Y	Y
TURBO BIT		Y(3)	Y(3)	Y	Y	Y	Y	Y	Y

Notes:

1. Dual-feedback on Global Macrocells.
2. These are global p-terms.
3. 7.5 ns version of 85C220 and 85C224 do not contain a Turbo-Bit.
4. 85C22V10 includes a clock invert option for each macrocell.

The 85C220 µPLD is a high-performance CMOS, pin-compatible, functional *superset* of 20-pin 16R8/R6/R4/L8/V8 PALs/GALs. Figure 7-1 shows the architecture of the 85C220 µPLD. Its 10 inputs and 8 I/O macrocells allow it to perform the same functions as 20-pin PALs/GALs. Its universal feedback from all 8 I/O pins makes it a *superset* of 16L8 and 16V8 devices, which do not allow feedback on the first and last I/O pins.

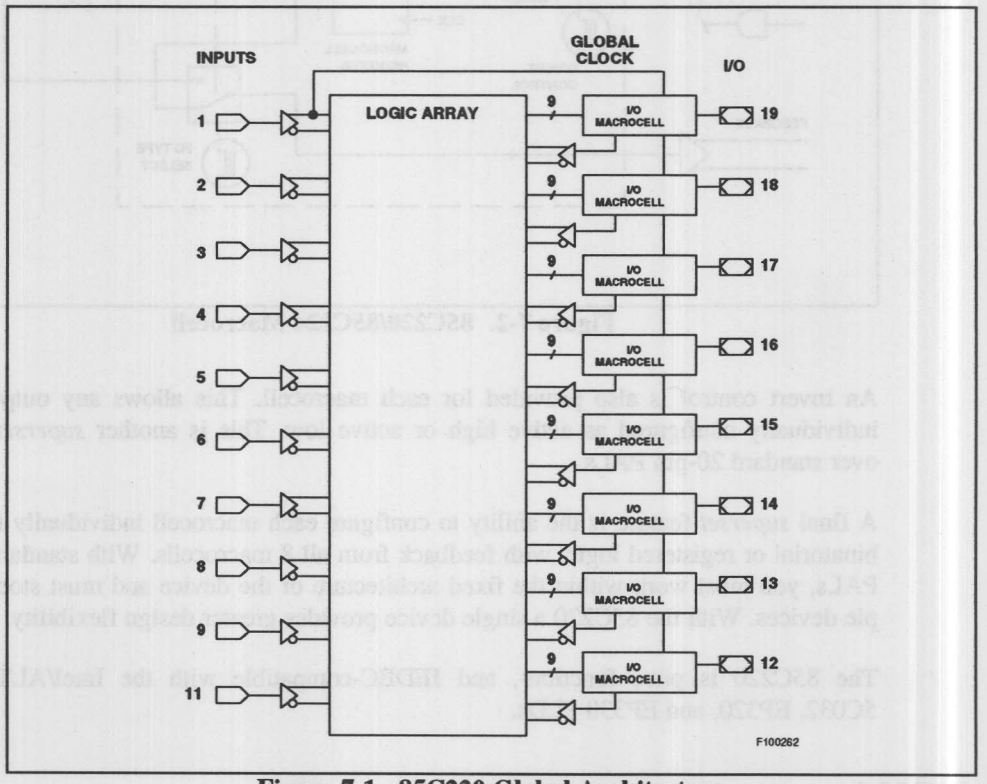


Figure 7-1. 85C220 Global Architecture

Figure 7-2 shows the macrocell architecture of the 85C220 µPLD. Note that there are 8 p-terms available at all times to the Sum-of-Products input. This is a *superset* feature over 16L8s and 16V8 combinatorial macrocells which only provide 7 p-terms (the 8th p-term on those devices is used for the OE control).

The 85C220 provides an additional *superset* feature in the form of a separate OE control (a 9th p-term) that is available at all times. With standard 20-pin registered PALs, a global OE control is provided, but designers cannot independently enable/disable register output buffers. GALs operate in the same way, using a global OE for all registers in the device. But with the 85C220, designers have independent access to all output buffers.

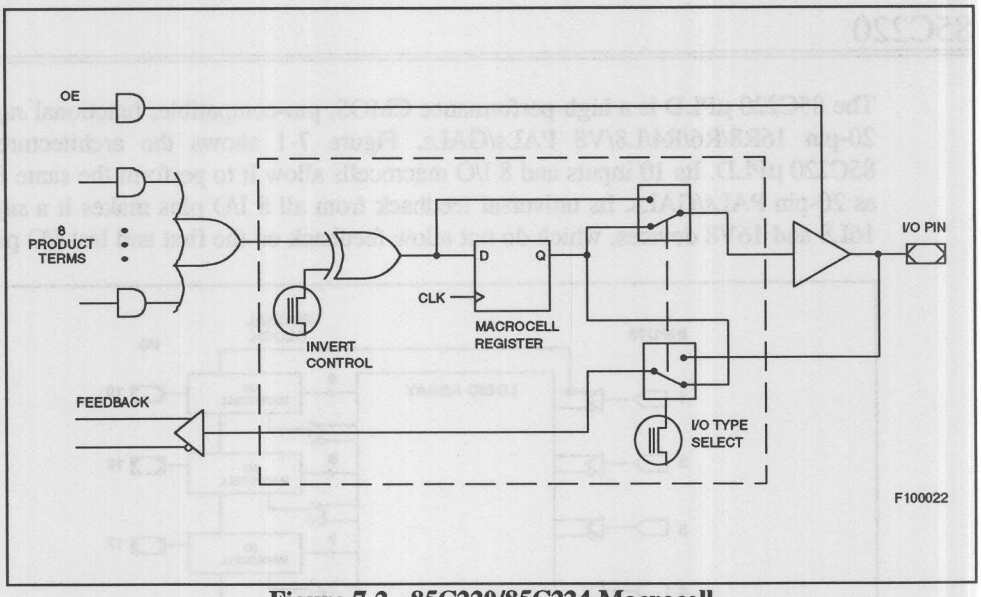


Figure 7-2. 85C220/85C224 Macrocell

An invert control is also provided for each macrocell. This allows any output to be individually configured as active high or active low. This is another *superset* feature over standard 20-pin PALs.

A final *superset* feature is the ability to configure each macrocell individually for combinatorial or registered logic, with feedback from all 8 macrocells. With standard 20-pin PALs, you must work within the fixed architecture of the device and must stock multiple devices. With the 85C220 a single device provides greater design flexibility.

The 85C220 is pin-, function-, and JEDEC-compatible with the Intel/ALTERA/TI 5C032, EP320, and EP330 PLDs.

85C224

The 85C224 µPLD is high-performance, pin-compatible, functional *superset* of 24-pin 20R8/R6/R4/L8/V8 PALs/GALs. Figure 7-3 shows the architecture of the 85C224 µPLD. Its 14 inputs and 8 I/O macrocells allow it to perform the same functions as 24-pin PALs/GALs. Like the 85C220, it has universal feedback from all 8 macrocells.

The macrocell architecture of the 85C224 µPLD is identical to the 85C220 (see Figure 7-2). Thus the compatibility with PALs/GALS is the same; so are the following *superset* features:

- 8 p-terms available at all times to the Sum-of-Products input for each macrocell.
- Separate OE control p-term available at all times for individual OE control.

- Inversion control available for each macrocell allows independent configuration of each output as active high or active low.
- Each macrocell can be individually configured as a combinatorial or registered output, with feedback from all 8 macrocells.

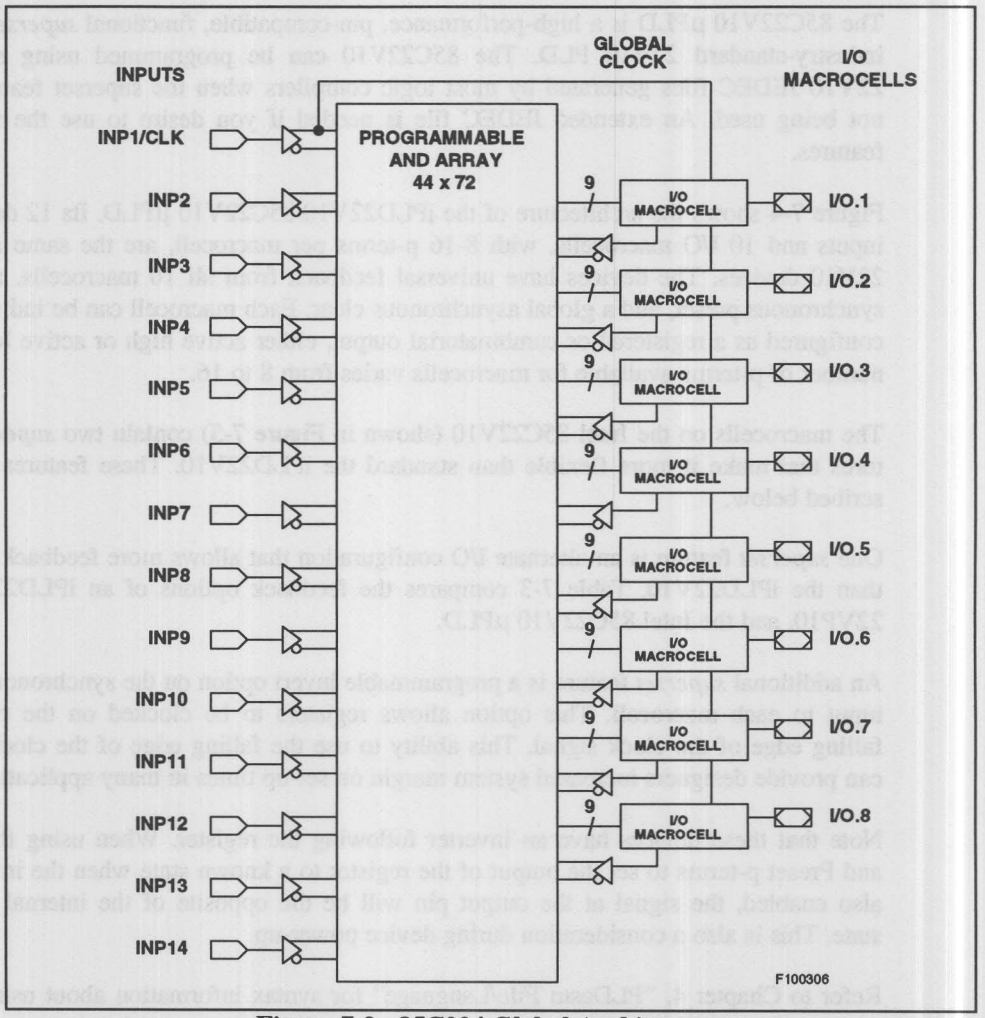


Figure 7-3. 85C224 Global Architecture

iPLD22V10/85C22V10

The iPLD22V10 µPLD is a high-performance, pin-compatible, CMOS version of the industry-standard 22V10 PLD. The iPLD22V10 can be programmed using standard 22V10 JEDEC files generated by most logic compilers.

The 85C22V10 µPLD is a high-performance, pin-compatible, functional *superset* of the industry-standard 22V10 PLD. The 85C22V10 can be programmed using standard 22V10 JEDEC files generated by most logic compilers when the superset features are not being used. An extended JEDEC file is needed if you desire to use the superset features.

Figure 7-4 shows the architecture of the iPLD22V10/85C22V10 µPLD. Its 12 dedicated inputs and 10 I/O macrocells, with 8-16 p-terms per macrocell, are the same as other 22V10 devices. The devices have universal feedback from all 10 macrocells, a global synchronous preset, and a global asynchronous clear. Each macrocell can be individually configured as a registered or combinatorial output, either active high or active low. The number of p-terms available for macrocells varies from 8 to 16.

The macrocells on the Intel 85C22V10 (shown in Figure 7-5) contain two *superset* features that make it more flexible than standard the iPLD22V10. These features are described below.

One *superset* feature is an alternate I/O configuration that allows more feedback options than the iPLD22V10. Table 7-3 compares the feedback options of an iPLD22V10, a 22VP10, and the Intel 85C22V10 µPLD.

An additional *superset* feature is a programmable invert option on the synchronous clock input to each macrocell. This option allows registers to be clocked on the rising or falling edge of the clock signal. This ability to use the falling edge of the clock signal can provide designers increased system margin on set-up times in many applications.

Note that these devices have an inverter following the register. When using the Clear and Preset p-terms to set the output of the register to a known state when the inverter is also enabled, the signal at the output pin will be the opposite of the internal register state. This is also a consideration during device power-up.

Refer to Chapter 4, "PLDasm File/Language" for syntax information about using these superset features.

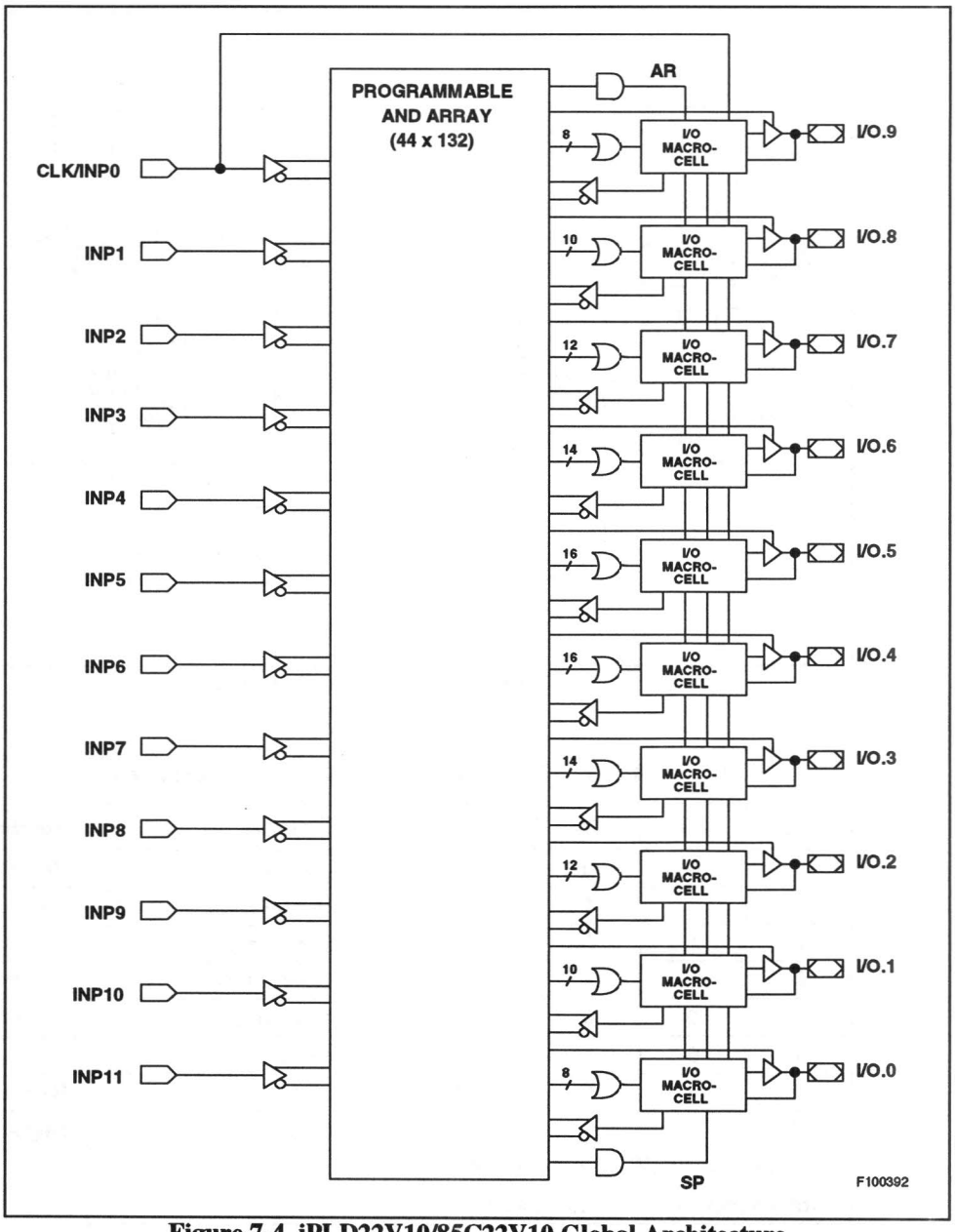


Figure 7-4. iPLD22V10/85C22V10 Global Architecture

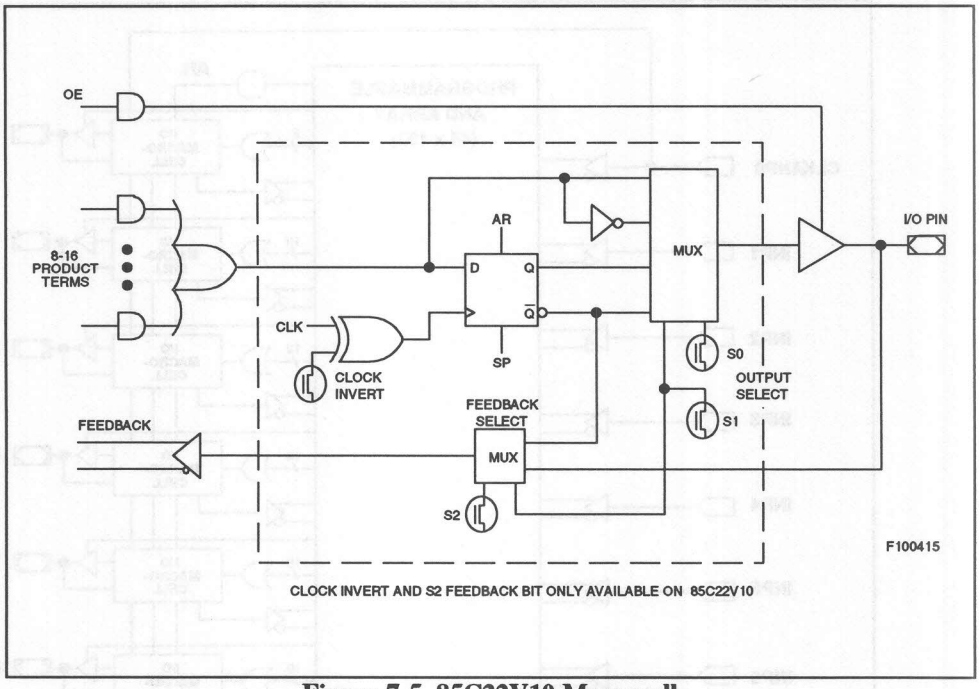


Figure 7-5. 85C22V10 Macrocell

Table 7-3. 85C22V10 Macrocell Configurations

S2	S1	S0	Output/Polarity	Feedback
0	0	0	Registered/Active Low	Registered
0	0	1	Registered/Active High	Registered
0	1	0	Combinatorial/Active Low	Pin
0	1	1	Combinatorial/Active High	Pin
1	0	0	**Registered/Active Low	Pin
1	0	1	**Registered/Active High	Pin
1	1	0	*Combinatorial/Active Low	Registered
1	1	1	*Combinatorial/Active High	Registered

*Not available on iPLD22V10 or 22VP10

**Not available on the iPLD22V10

iPLD610/85C060

The iPLD610 and 85C060 are a high-performance, high I/O count µPLDs that are speed upgrades to the following devices:

- Intel/ALTERA 5C060/EP600 PLDs
- ALTERA/TI EP610 and EP630 PLDs
- AMD PALCE630 PLDs.

The iPLD610 and 85C060 are the fastest members of this *industry-standard* architecture and are pin-, function-, and JEDEC-compatible with this family of devices.

As shown in Figure 7-6, these devices feature 16 I/O macrocells in 24-pin DIP and 28-pin PLCC packages. Four dedicated inputs and two dedicated clocks (one clock for each bank of 8 registers) complete the device architecture. CLK1 clocks the registers on half the device; CLK2 clocks the registers on the other half.

Figure 7-7 shows the macrocell architecture. Note that 8 p-terms are available to the SOP input at all times. Each macrocell contains an inversion control bit that allows any output to be individually configured as active high or active low. Finally, the OE p-term on each macrocell can be used as an asynchronous clock when not needed to control the macrocell output buffer. This allows up to 16 asynchronous clocks to be generated internally via logic equations.

The output from each macrocell can be independently configured as combinatorial or registered. Four different register types are available for each macrocell: D-type, T-type, JK-type, and SR-type. This wide range of features makes the iPLD610/85C060 architecture ideally suited for register- and I/O-intensive designs. (JK- and SR-registers are emulations.)

Figures 7-8 shows the device clocking. Refer to Chapter 4, "PLDasm File/Language" for syntax information on clocking µPLDs. There are two synchronous clock inputs. Each clock input drives eight registers. If more than eight registers are connected to the same synchronous clock input, both clock pins are needed to implement the circuit. In this event, the pin diagram on the Utilization Report shows both pins as clocks with a message to tie both pins together.

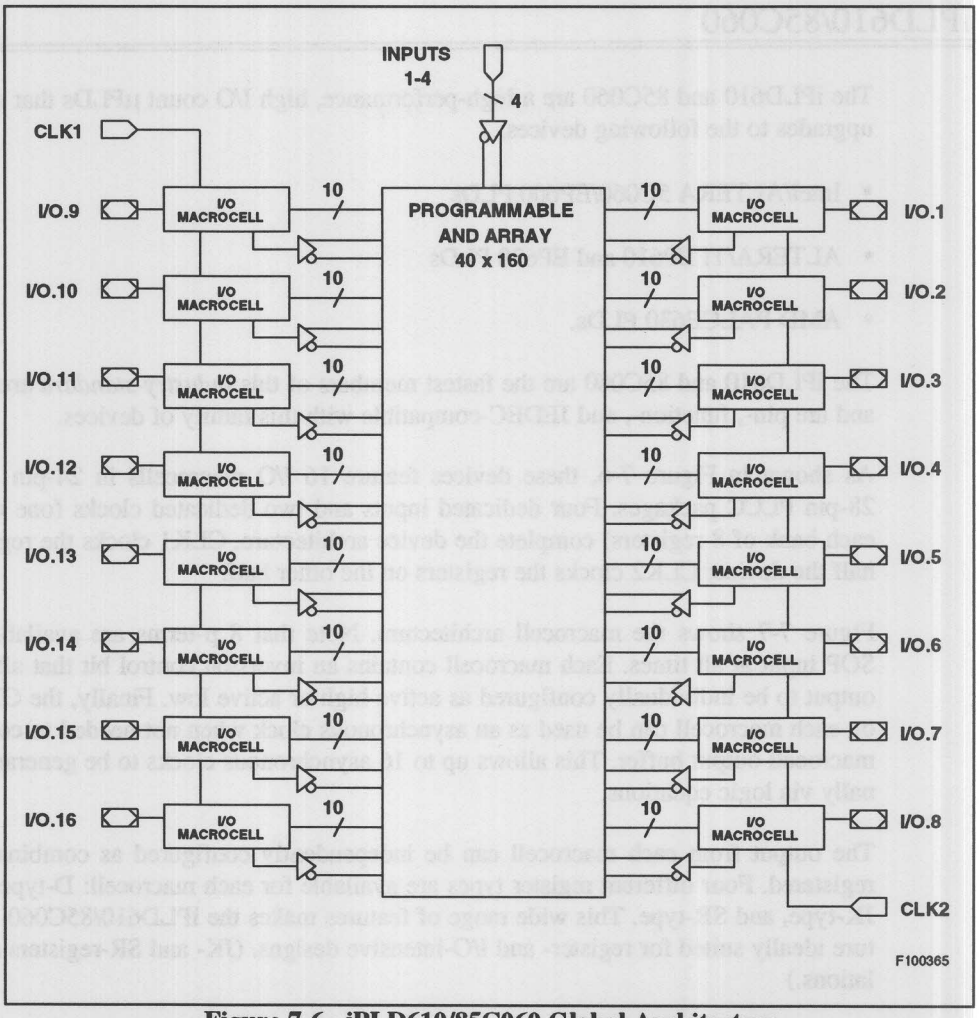


Figure 7-6. iPLD610/85C060 Global Architecture

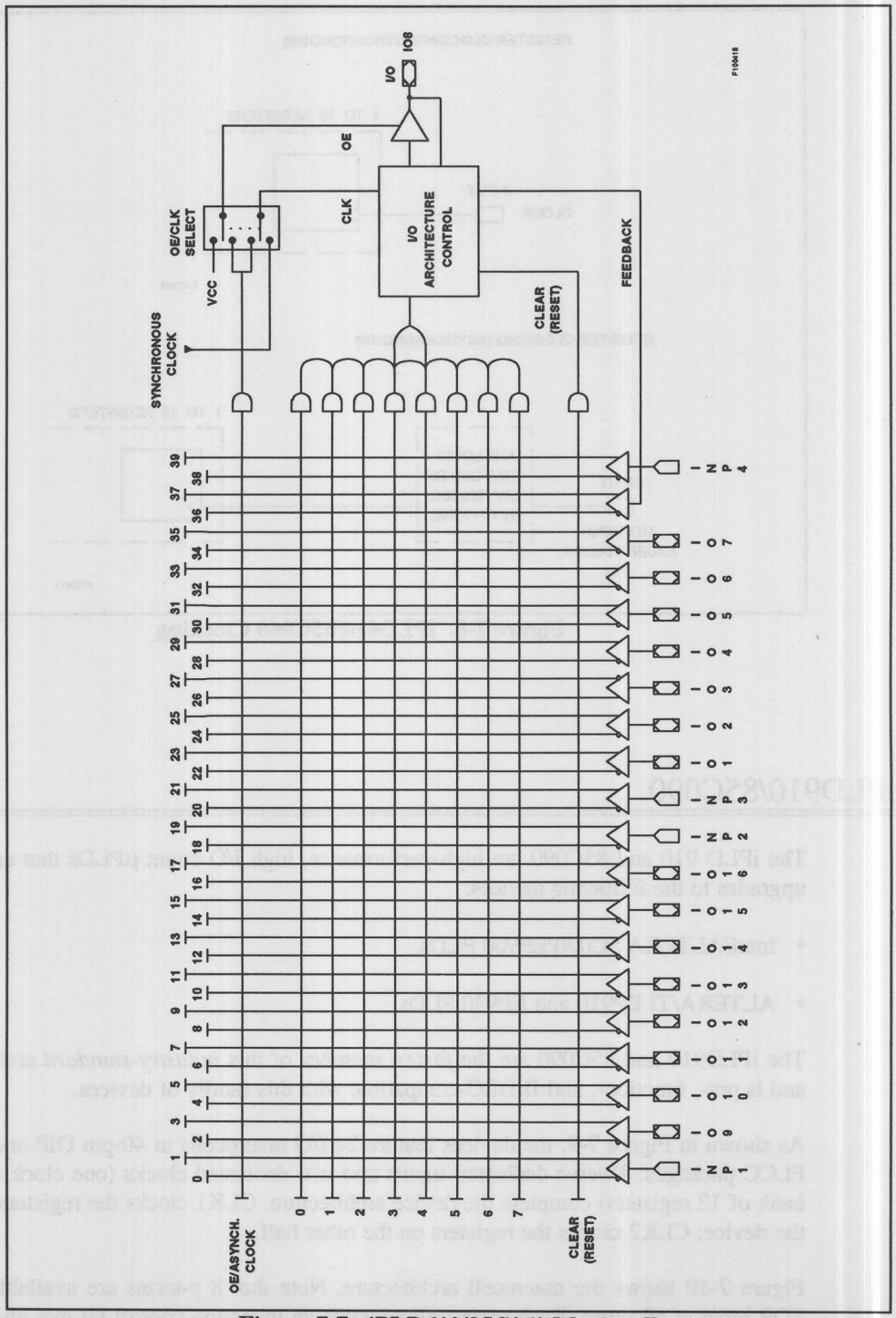


Figure 7-7. iPLD610/85C060 Macrocell

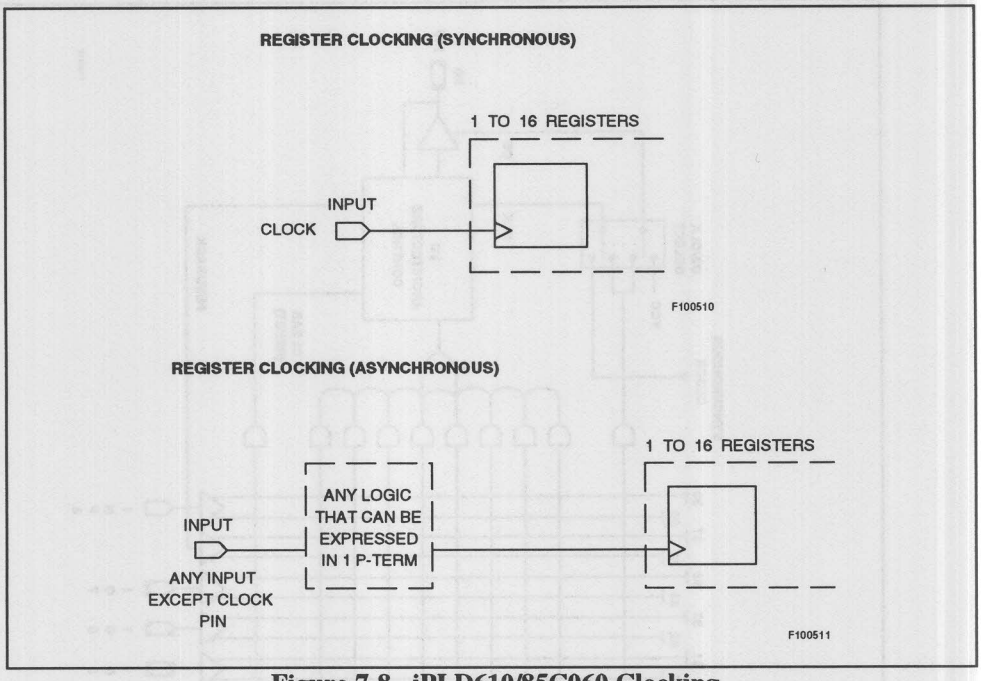


Figure 7-8. iPLD610/85C060 Clocking

iPLD910/85C090

The iPLD 910 and 85C090 are high-performance, high I/O count µPLDs that are speed upgrades to the following devices:

- Intel/ALTERA 5C090/EP900 PLDs
- ALTERA/TI EP910 and EP930 PLDs

The iPLD910 and 85C090 are the *fastest* member of this *industry-standard* architecture and is pin-, function-, and JEDEC-compatible with this family of devices.

As shown in Figure 7-9, the devices feature 24 I/O macrocells in 40-pin DIP and 44-pin PLCC packages. Twelve dedicated inputs and two dedicated clocks (one clock for each bank of 12 registers) complete the device architecture. CLK1 clocks the registers on half the device; CLK2 clocks the registers on the other half.

Figure 7-10 shows the macrocell architecture. Note that 8 p-terms are available to the SOP input at all times. Each macrocell contains an inversion control bit that allows any output to be individually configured as active high or active low. Finally, the OE p-term

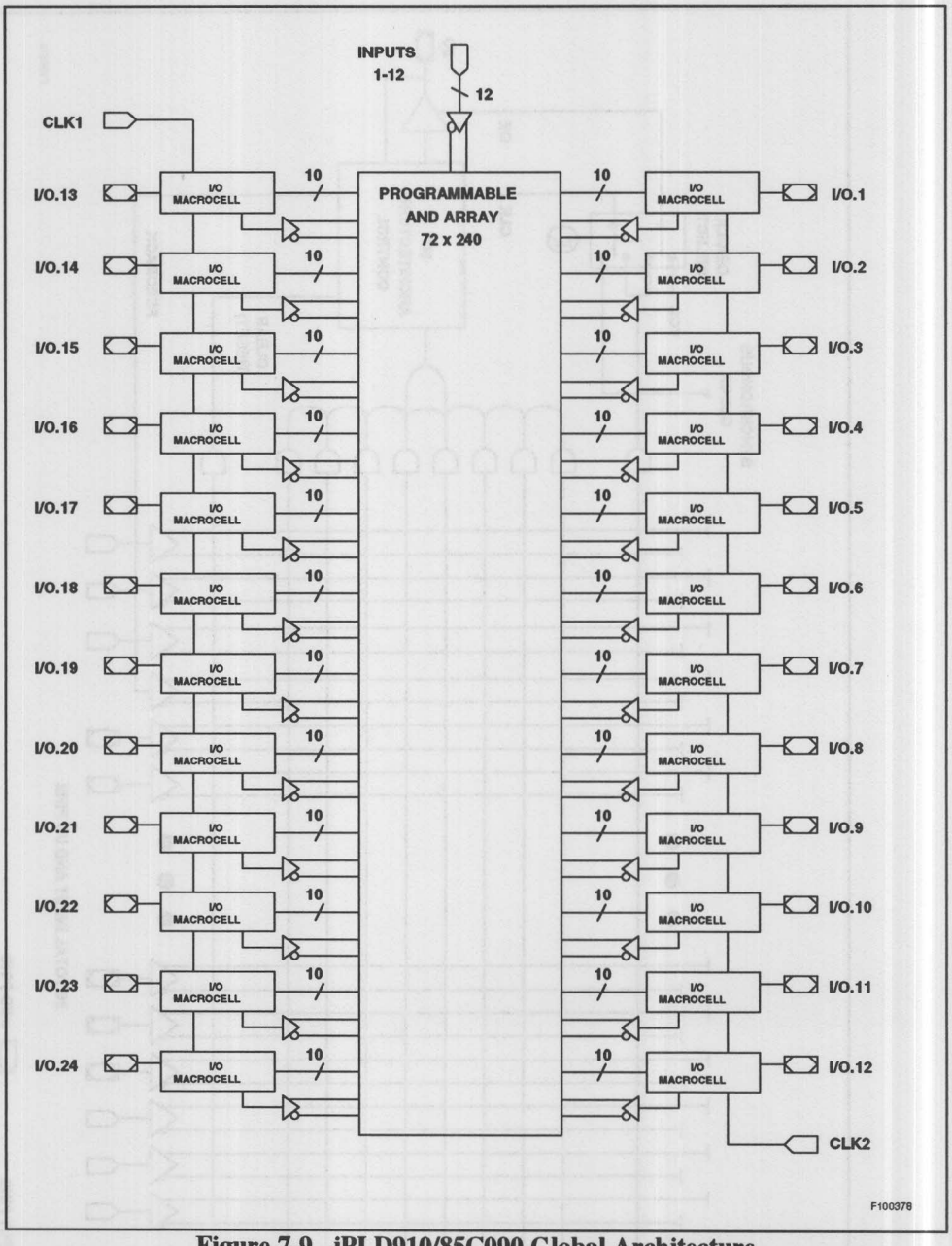


Figure 7-9. iPLD910/85C090 Global Architecture

on each macrocell can be used as an asynchronous clock when not needed to control the macrocell output buffer. This allows up to 24 asynchronous clocks to be generated internally via logic equations.

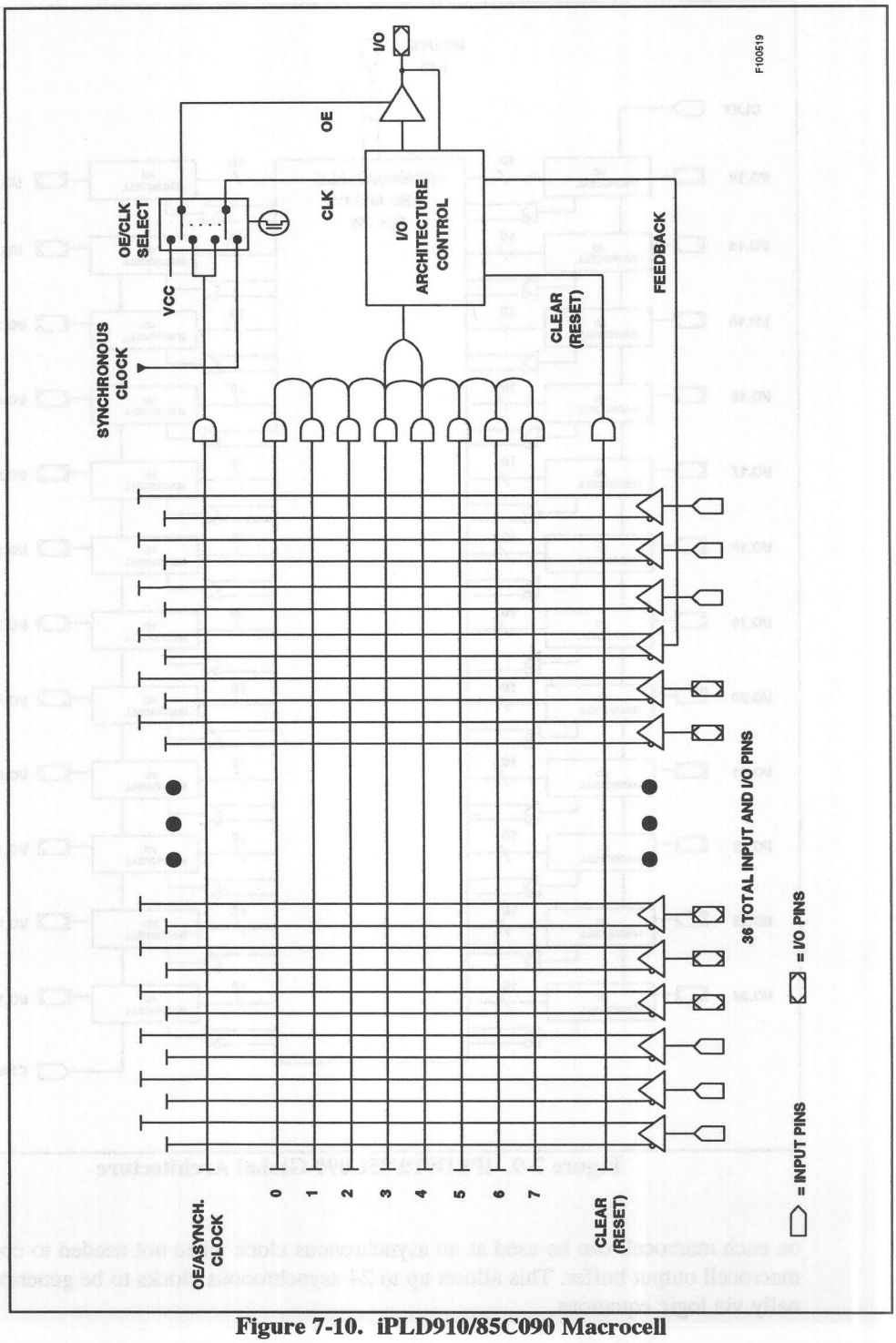


Figure 7-10. iPLD910/85C090 Macrocell

The output from each macrocell can be independently configured as combinatorial or registered. Four different register types are available for each macrocell: D-type, T-type, JK-type, and SR-type. This wide range of features makes the 85C090 architecture ideally suited for register- and I/O-intensive designs. (JK- and SR-registers are emulations.)

Figure 7-11 shows the device clocking. Refer to Chapter 4, "PLDasm File/Language" for syntax information about clocking µPLDs. There are two synchronous clock inputs. Each clock input drives 12 registers. If more than 12 registers are connected to the same synchronous clock input, both clock pins are needed to implement the circuit. In this event, the pin diagram on the Utilization Report shows both pins as clocks with a message to tie both pins together.

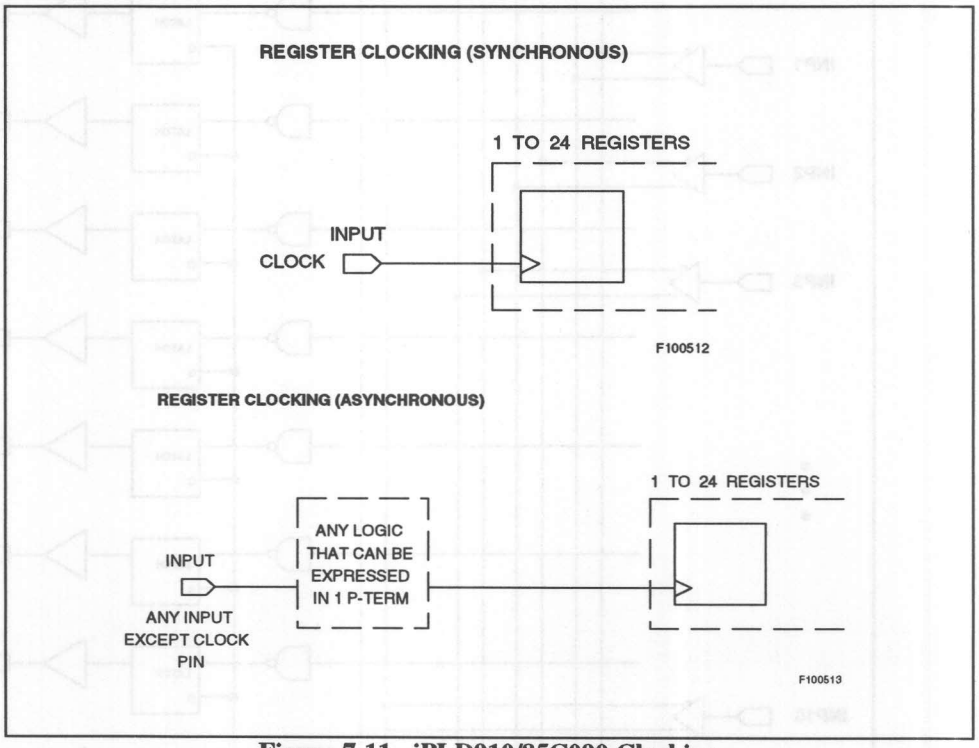


Figure 7-11. iPLD910/85C090 Clocking

85C508

The 85C508 is a high-speed µPLD designed for address decoding/latching in microcomputer systems. This device contains 16 dedicated inputs, 8 latched outputs and one global latch enable (LE) signal.

Figure 7-12 shows the global architecture of the 85C508. Each of the outputs is a transparent latch fed by a single NAND p-term.

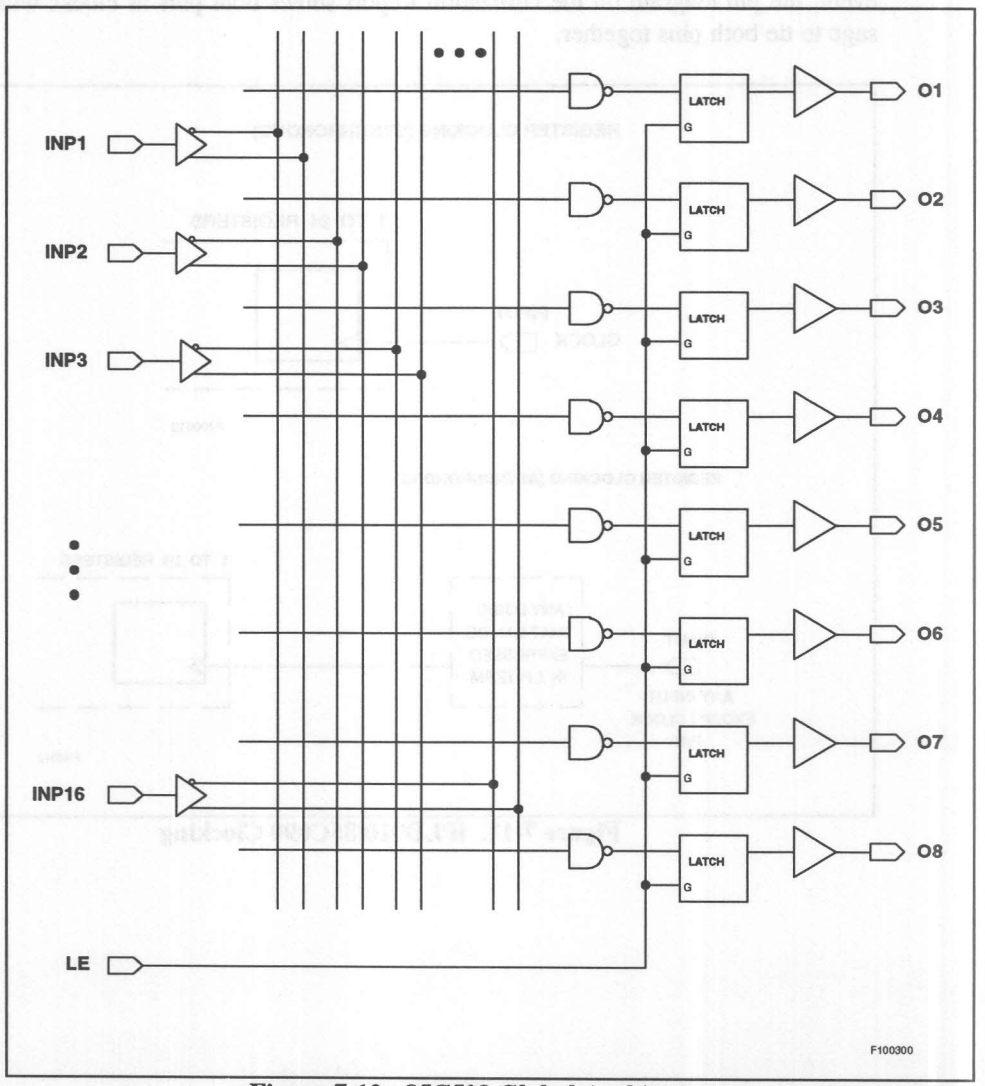


Figure 7-12. 85C508 Global Architecture

5AC312

The 5AC312 is a device that provides eight programmable inputs, 12 I/O pins, and two inputs that can serve as clocks or dedicated inputs (see Figure 7-13). Associated with each I/O pin is a macrocell that can be programmed to provide registered or combinatorial output. D, T, JK, or SR registers are supported (JK- and SR-registers are emulations). Each macrocell supports from 0 to 16 p-terms through a proprietary p-term allocation scheme. Each macrocell also supports 2 p-terms on its Clear, Preset, Output Enable, and Asynchronous Clock inputs. Registers can be clocked synchronously from the global clock (CLK).

The eight p-terms for each macrocell are organized as two groups of four p-terms each (see Figure 7-14). Each group of four p-terms can be allocated to an adjacent macrocell when not needed for the current macrocell. If more than eight p-terms are needed for a macrocell, p-terms can be allocated from adjacent macrocells. The 12 macrocells are organized into two rings of six macrocells. P-term allocation is supported within a ring. Allocation between the two rings is not supported. Table 7-4 lists the p-term allocation rings and the previous/next macrocells for each macrocell in the 5AC312.

Table 7-4. 5AC312 Previous/Next Macrocells

Ring 1			Ring 2		
Current Macrocell	Next Macrocell	Previous Macrocell	Current Macrocell	Next Macrocell	Previous Macrocell
1	2	6	7	8	12
2	3	1	8	9	7
3	4	2	9	10	8
4	5	3	10	11	9
5	6	4	11	12	10
6	1	5	12	7	11

Figure 7-15 shows the programmable inputs of the 5AC312. Each programmable input can be configured as a flow-through input, an input latch, or an input register. Inputs configured as a latch or register can be clocked by the global ILE/ICLK signal or asynchronously by a p-term from the logic array.

Figure 7-16 shows the device clocking for the 5AC312 and the 5AC324. Refer to Chapter 4, "PLDasm File/Language" for syntax information on clocking µPLDs.

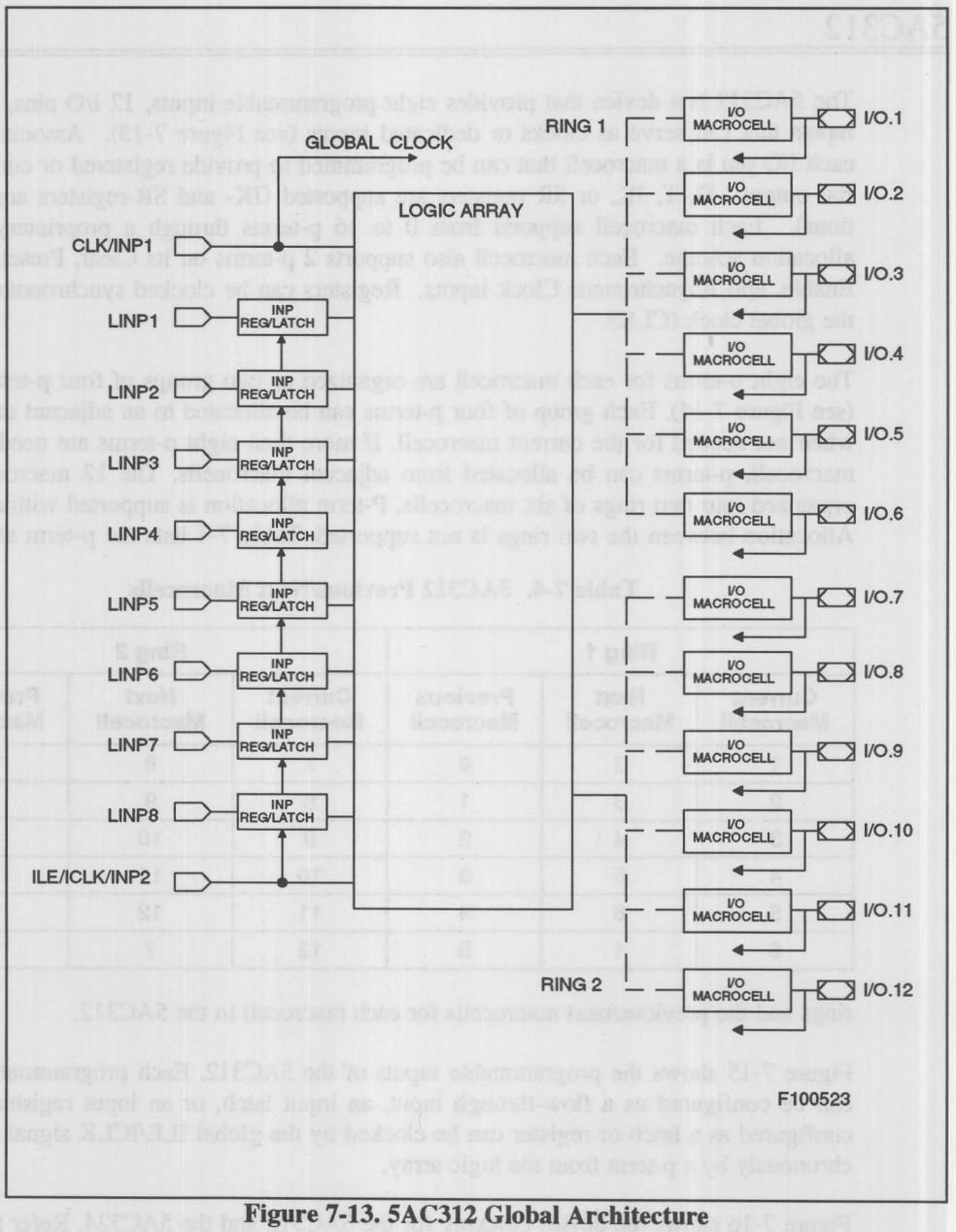


Figure 7-13. 5AC312 Global Architecture

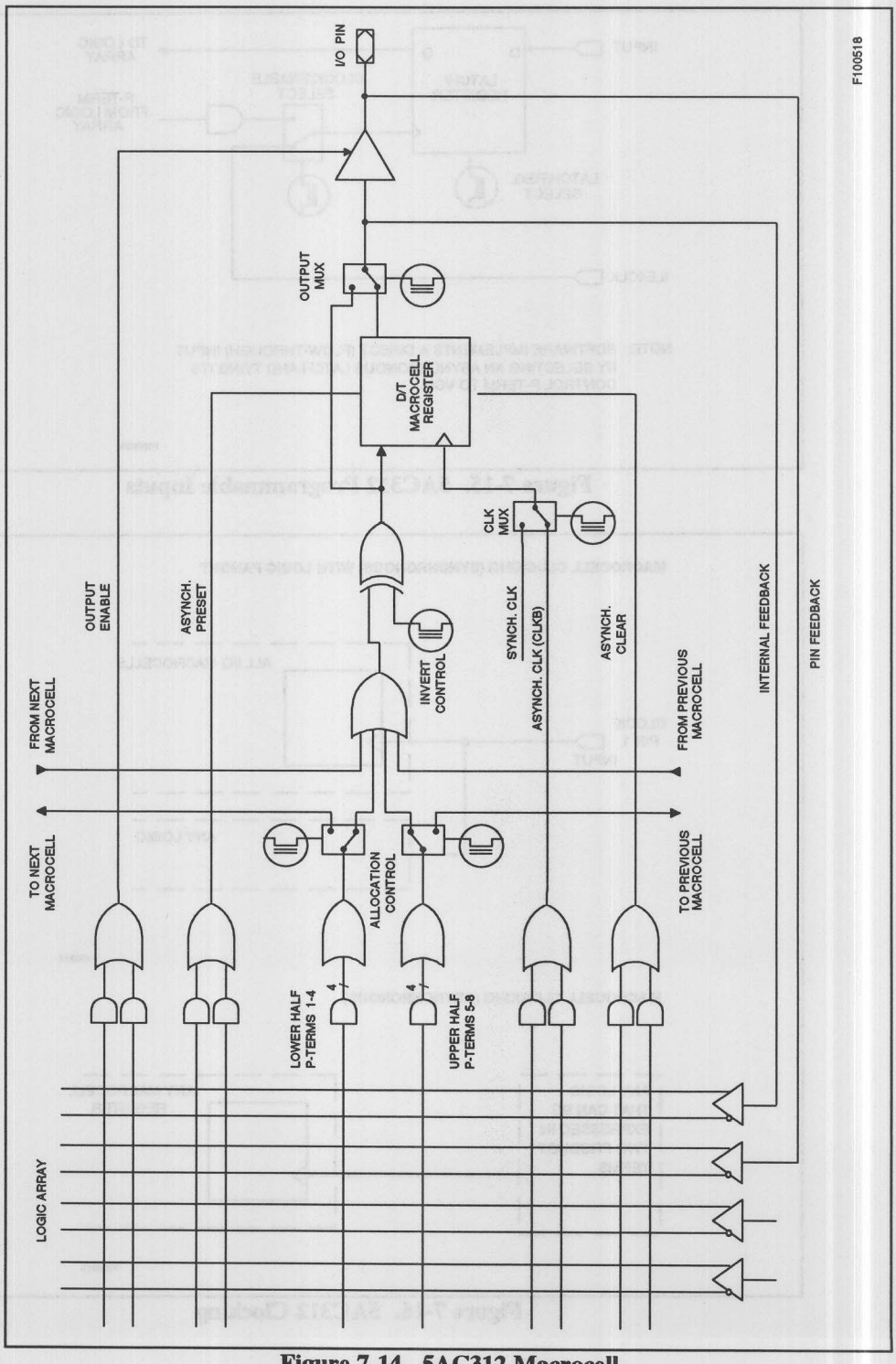


Figure 7-14. 5AC312 Macrocell

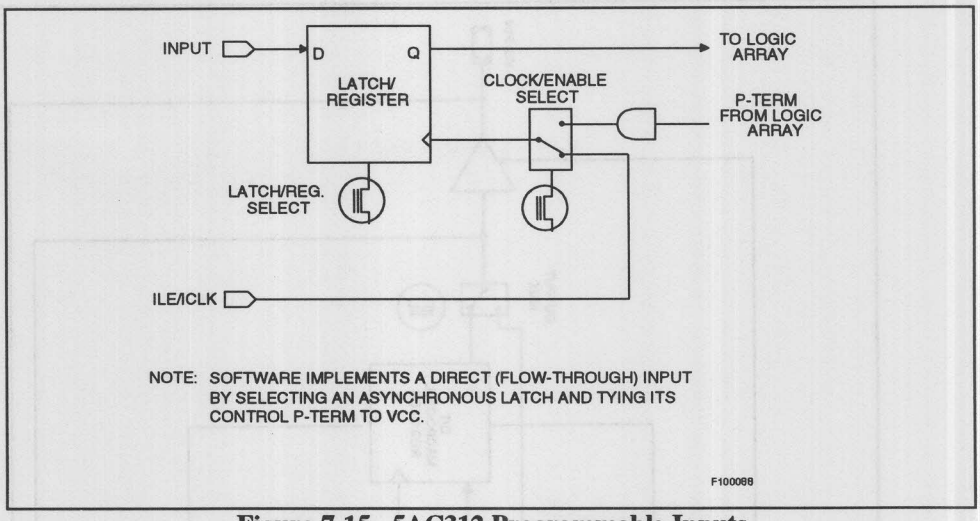


Figure 7-15. 5AC312 Programmable Inputs

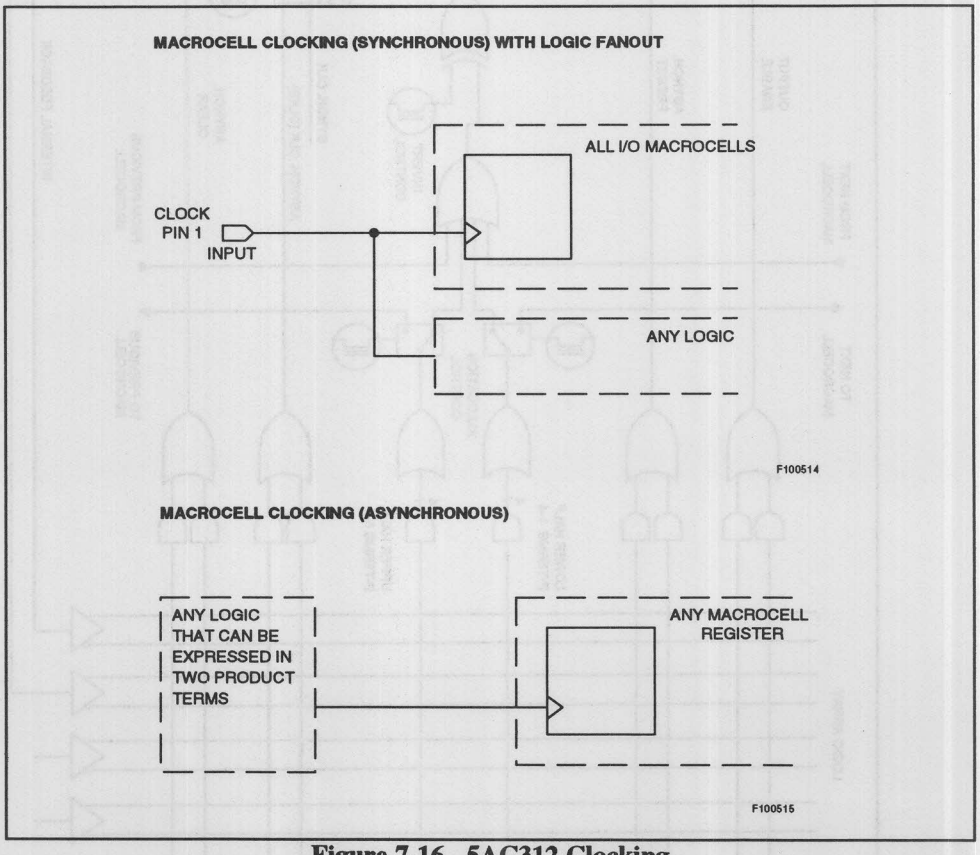


Figure 7-16. 5AC312 Clocking

5AC324

The 5AC324 is a device that provides 10 programmable inputs, 24 I/O pins, and two inputs that can serve as clocks or dedicated inputs (see Figure 7-17). Associated with each I/O pin is a macrocell that can be programmed to provide registered or combinatorial output. D, T, JK, or SR registers are supported (JK- and SR-registers are emulations). Each macrocell supports from 0 to 16 p-terms through a proprietary p-term allocation scheme. Each macrocell also supports 2 p-terms on its Clear, Preset, Output Enable, and Asynchronous Clock inputs. Registers can be clocked synchronously from the global clock (CLK).

The eight p-terms for each macrocell are organized as two groups of four p-terms each (see Figure 7-18). Each group of four p-terms can be allocated to an adjacent macrocell when not needed for the current macrocell. If more than eight p-terms are needed for a macrocell, p-terms can be allocated from adjacent macrocells. The 24 macrocells are organized into two rings of 12 macrocells. P-term allocation is supported within a ring. Allocation between the two rings is not supported. Table 7-5 lists the p-term allocation rings and the previous/next macrocells for each macrocell in the 5AC324.

Figure 7-19 shows the programmable inputs of the 5AC324. Each programmable input can be configured as a flow-through input, an input latch, or an input register. Inputs configured as a latch or register can be clocked by the global ILE/ICLK signal or asynchronously by a p-term from the logic array.

Refer to Figure 7-20 for the device clocking. Refer to Chapter 4, "PLDasm File/Language" for syntax information on clocking µPLDs.

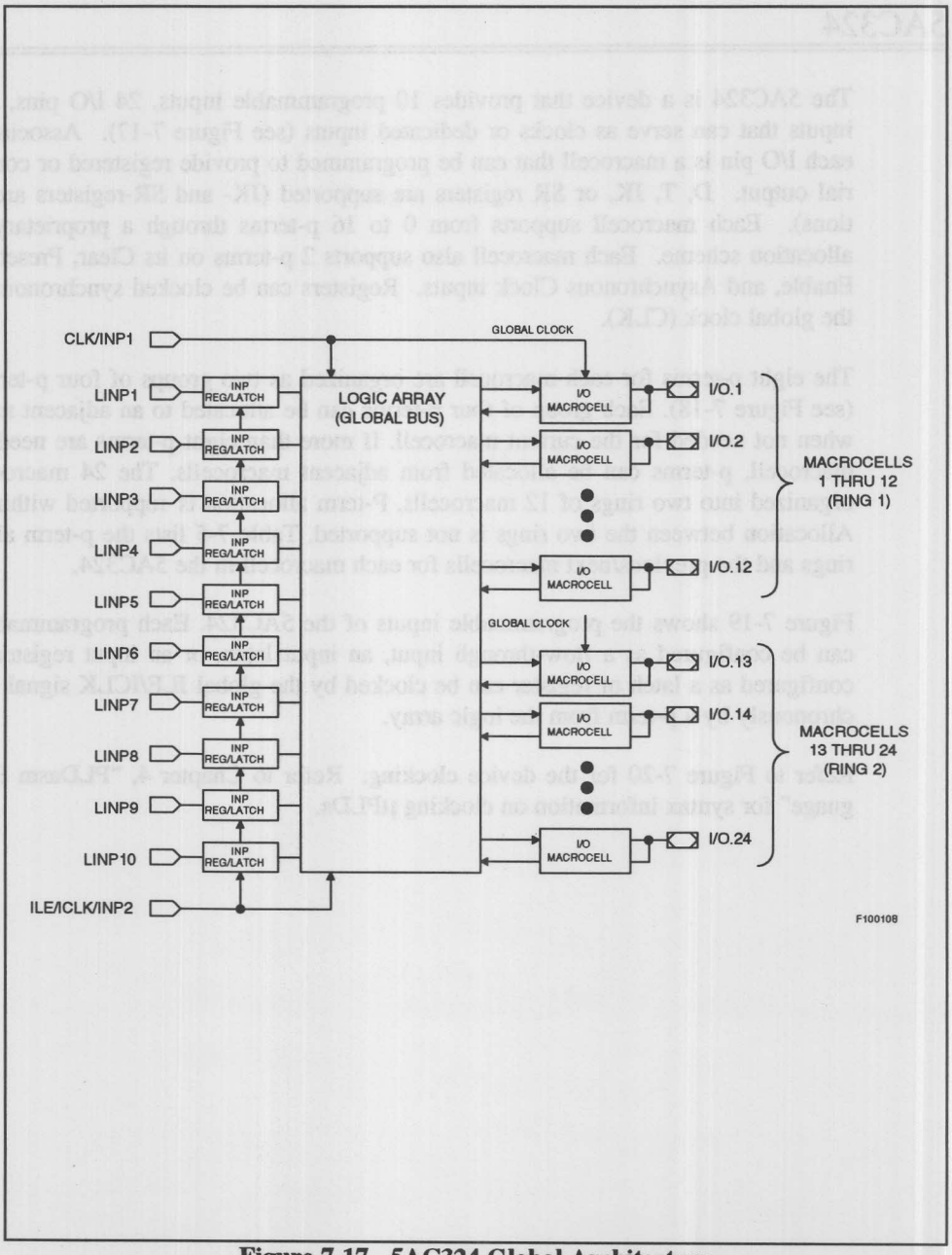


Figure 7-17. 5AC324 Global Architecture

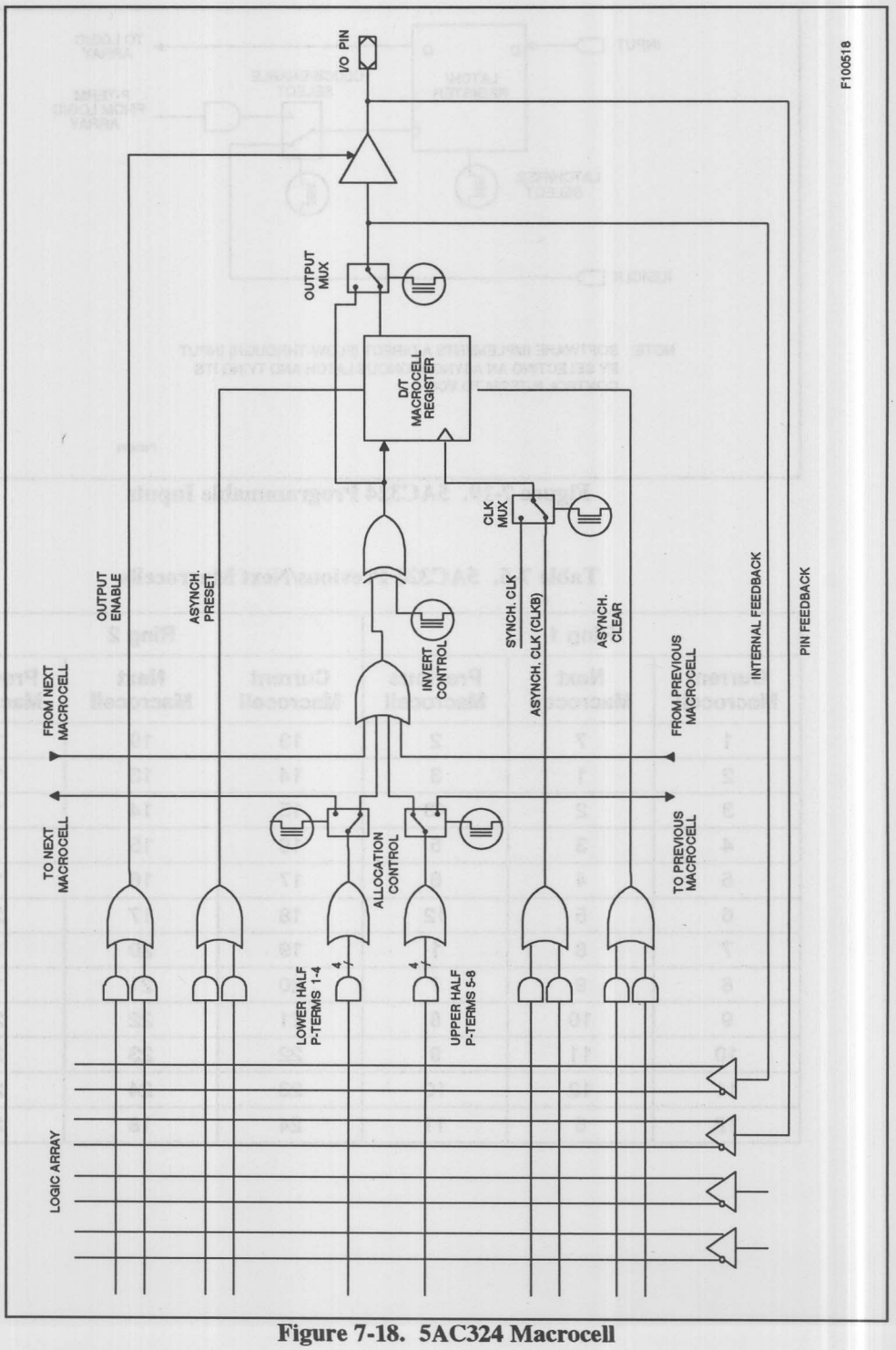


Figure 7-18. 5AC324 Macrocell

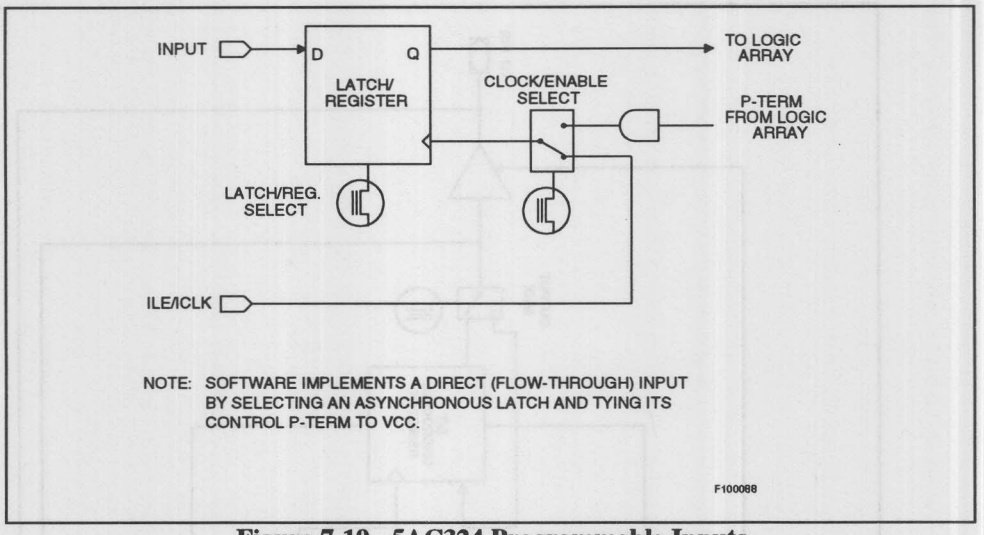


Figure 7-19. 5AC324 Programmable Inputs

Table 7-5. 5AC324 Previous/Next Macrocells

Ring 1			Ring 2		
Current Macrocell	Next Macrocell	Previous Macrocell	Current Macrocell	Next Macrocell	Previous Macrocell
1	7	2	13	19	14
2	1	3	14	13	15
3	2	43	15	14	16
4	3	5	16	15	17
5	4	6	17	16	18
6	5	12	18	17	24
7	8	1	19	20	13
8	9	7	20	21	19
9	10	8	21	22	20
10	11	9	22	23	21
11	12	10	23	24	22
12	6	11	24	18	23

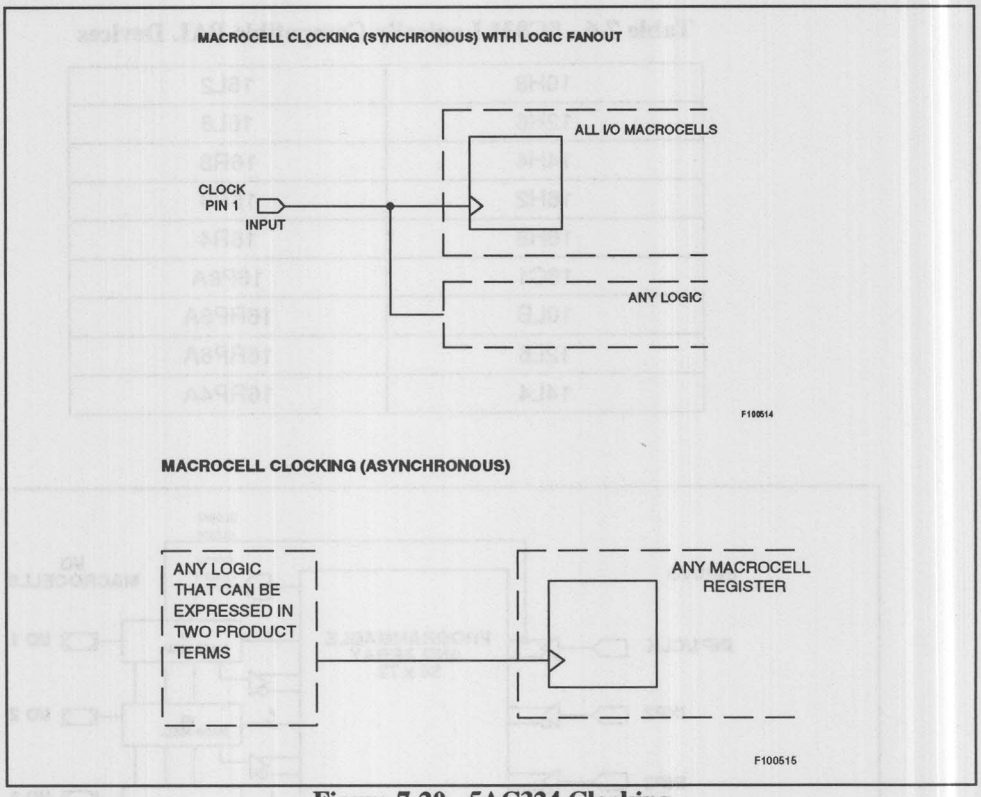


Figure 7-20. 5AC324 Clocking

5C031

The 5C031 is a logical superset of many 20-pin bipolar PAL devices. The I/O and logic sections of the 5C031 device can be configured to emulate devices listed in Table 7-6.

As shown in Figure 7-21, the 5C031 contains 10 dedicated inputs and eight I/O pins. These I/O pins can be individually configured to be inputs, outputs or bidirectional I/O pins. Each of these I/O pins is connected to a macrocell. The 5C031 contains eight identical macrocells.

Each macrocell (see Figure 7-22) consists of a PLA (programmable logic array) block and an I/O architecture block. The I/O architecture block contains a D-type register. The PLA block consists of eight 36-input AND gates feeding into an OR gate. The output of this PLA block is fed into the architecture control block (Figure 7-23).

Figure 7-24 shows clocking for the devices. Refer to Chapter 4, "PLDasm File/Language" for syntax information about clocking µPLDs. The 5C031 also provides a global Asynchronous Clear and a global Synchronous Preset p-term. Note that this device has an inverter following the register. When using the Clear and Preset p-terms to set the

Table 7-6. 5C031 Logically Compatible PAL Devices

10H8	16L2
12H6	16L8
14H4	16R8
16H2	16R6
16H8	16R4
16C1	16P8A
10LB	16RP8A
12L6	16RP6A
14L4	16RP4A

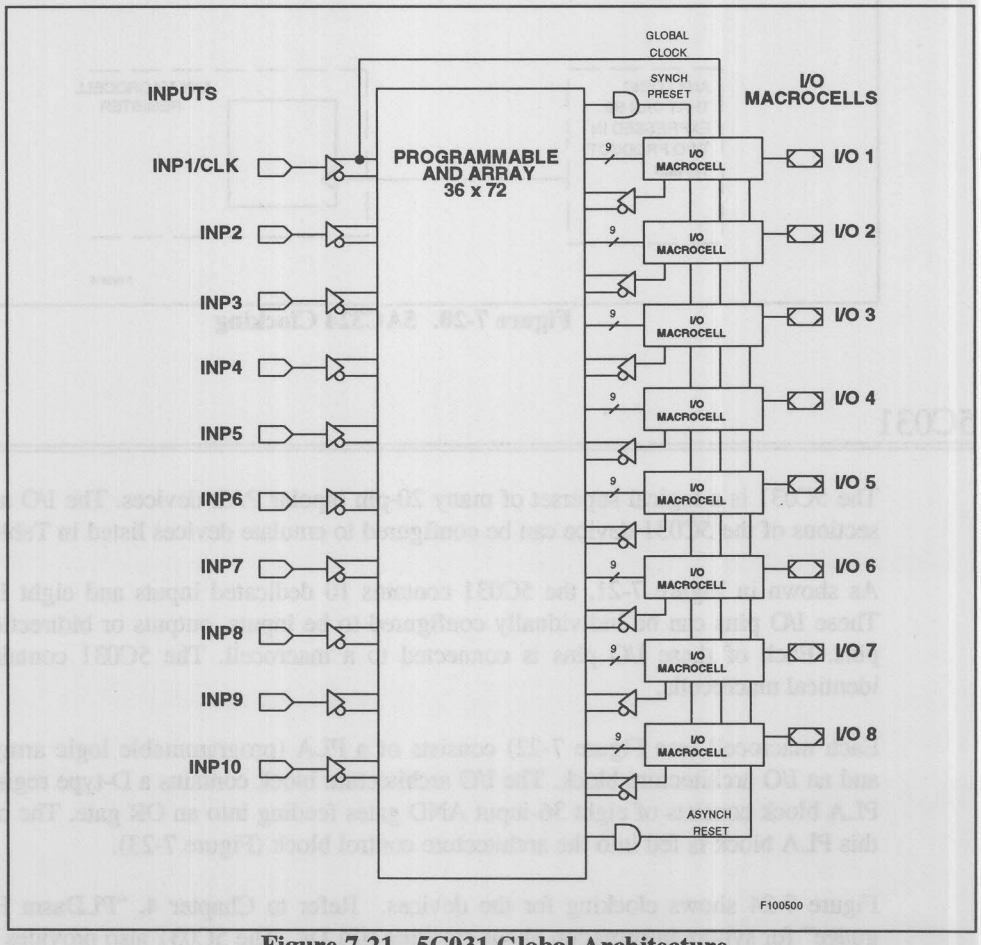


Figure 7-21. 5C031 Global Architecture

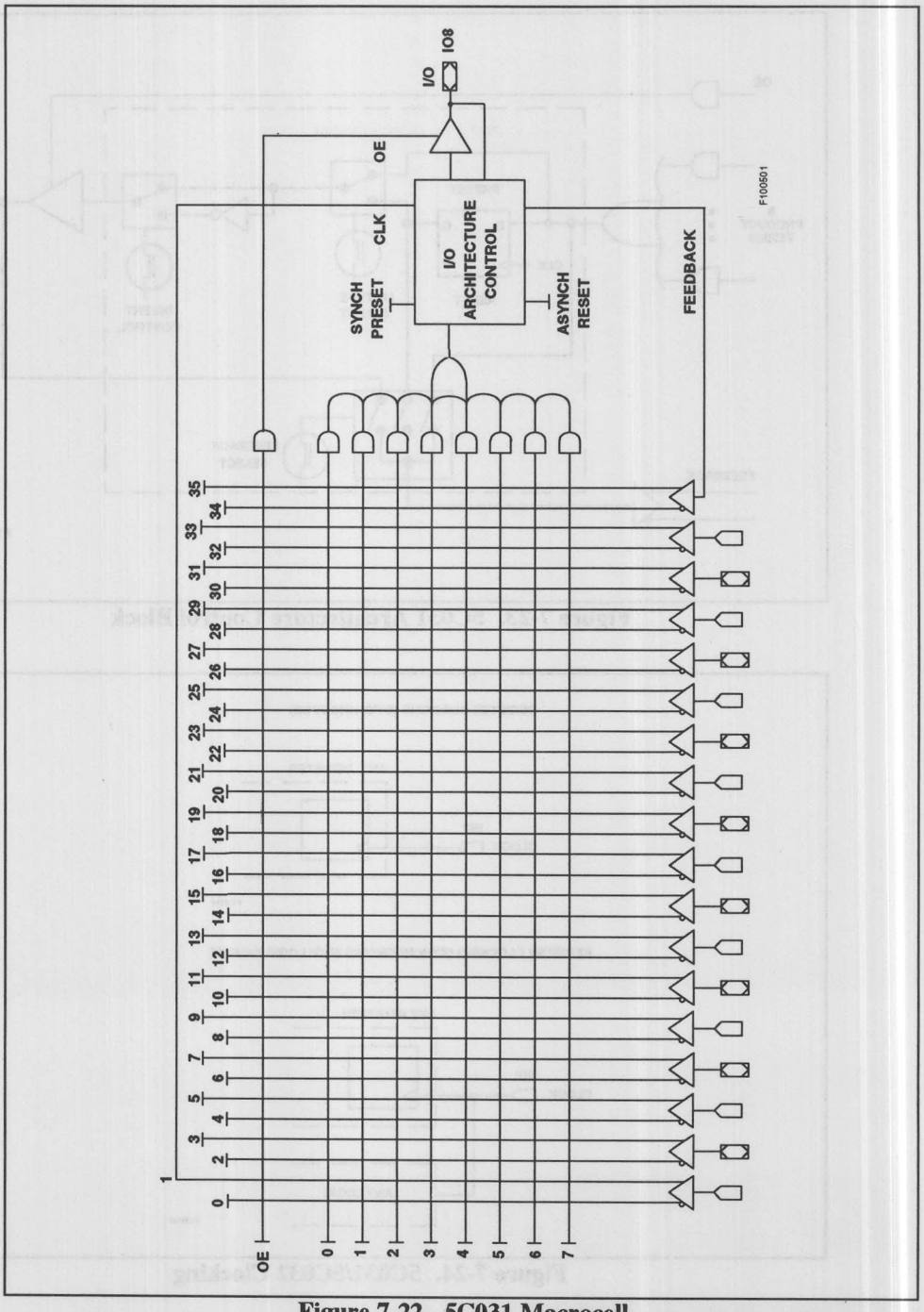


Figure 7-22. 5C031 Macrocell

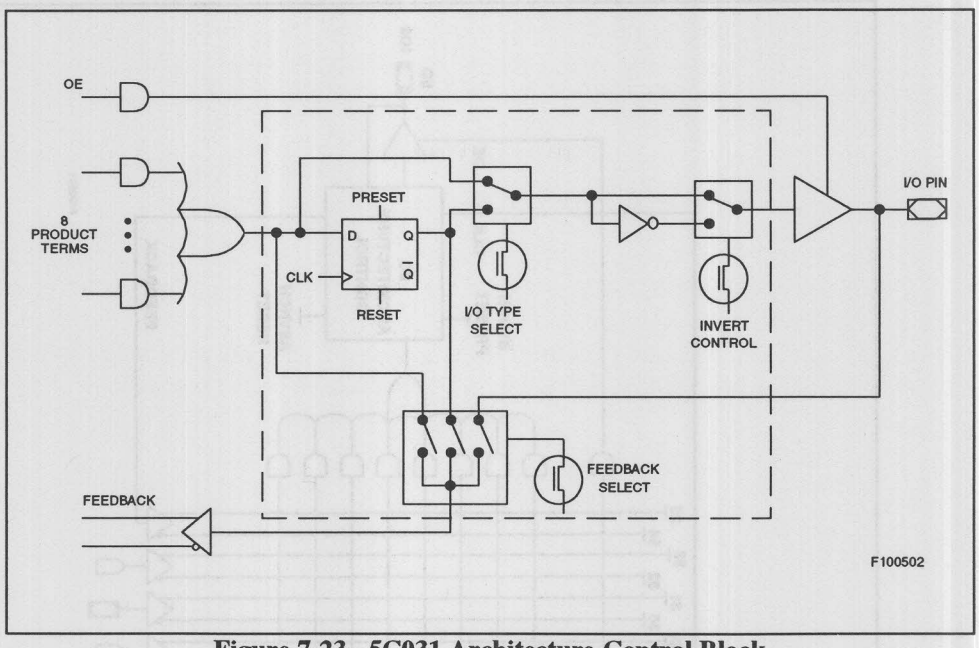


Figure 7-23. 5C031 Architecture Control Block

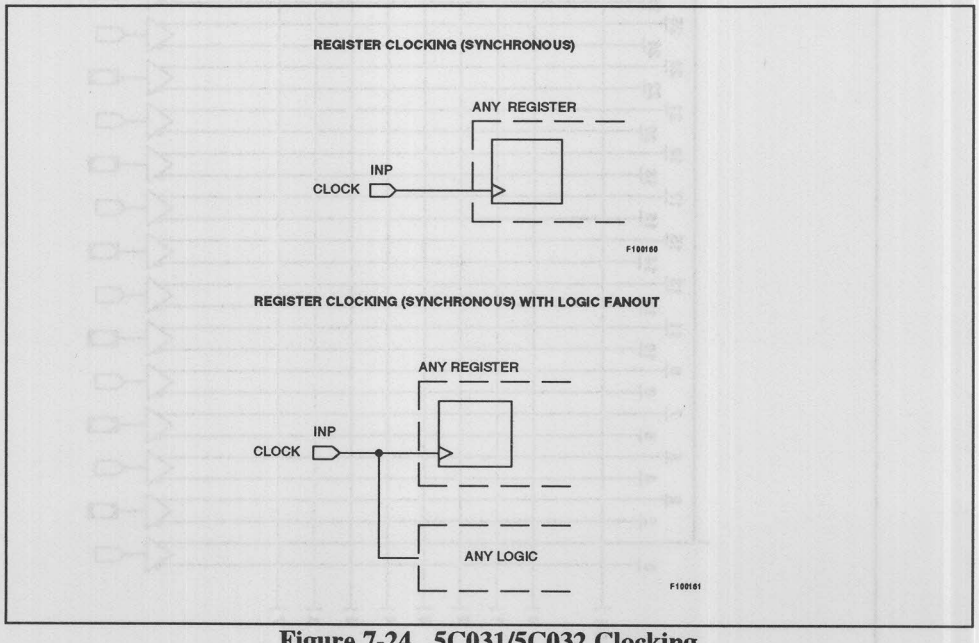


Figure 7-24. 5C031/5C032 Clocking

output of registers to a known state while also using the inverter, the signal at the output pin will be the opposite of the internal register state. This is also a consideration during device power-up.

5C032

The 5C032 µPLD is a high-performance CMOS, pin-compatible, functional *superset* of 20-pin 16R8/R6/R4/L8/V8 PALs/GALs. Figure 7-25 shows the architecture of the 5C032 µPLD. Its 10 inputs and 8 I/O macrocells allow it to perform the same functions as 20-pin PALs/GALs. Its universal feedback from all 8 I/O pins makes it a *superset* of 16L8 and 16V8 devices, which do not allow feedback on the first and last I/O pins.

Figure 7-26 shows the macrocell architecture of the 5C032 µPLD. Note that there are 8 p-terms available at all times to the Sum-of-Products input. This is a *superset* feature over 16L8s and 16V8 combinatorial macrocells which only provide 7 p-terms (the 8th p-term on those devices is used for the OE control).

The 5C032 provides an additional *superset* feature in the form of a separate OE control (a 9th p-term) that is available at all times. With standard 20-pin registered PALs, a global OE control is provided, but designers cannot independently enable/disable register output buffers. GALs operate in the same way, using a global OE for all registers in the device. But with the 5C032, designers have independent access to all output buffers.

An invert control is also provided for each macrocell. This allows any output to be individually configured as active high or active low. This is another *superset* feature over standard 20-pin PALs.

A final *superset* feature is the ability to configure each macrocell individually for combinatorial or registered logic, with feedback from all 8 macrocells. With standard 20-pin PALs, you must work within the fixed architecture of the device and must stock multiple devices. With the 5C032, a single device provides greater design flexibility.

The 5C032 is pin-, function-, and JEDEC-compatible with the Intel/ALTERA/TI 5C032, EP320, and EP330 PLDs.

NOTE

For a higher speed version of this architecture, see the 85C220.

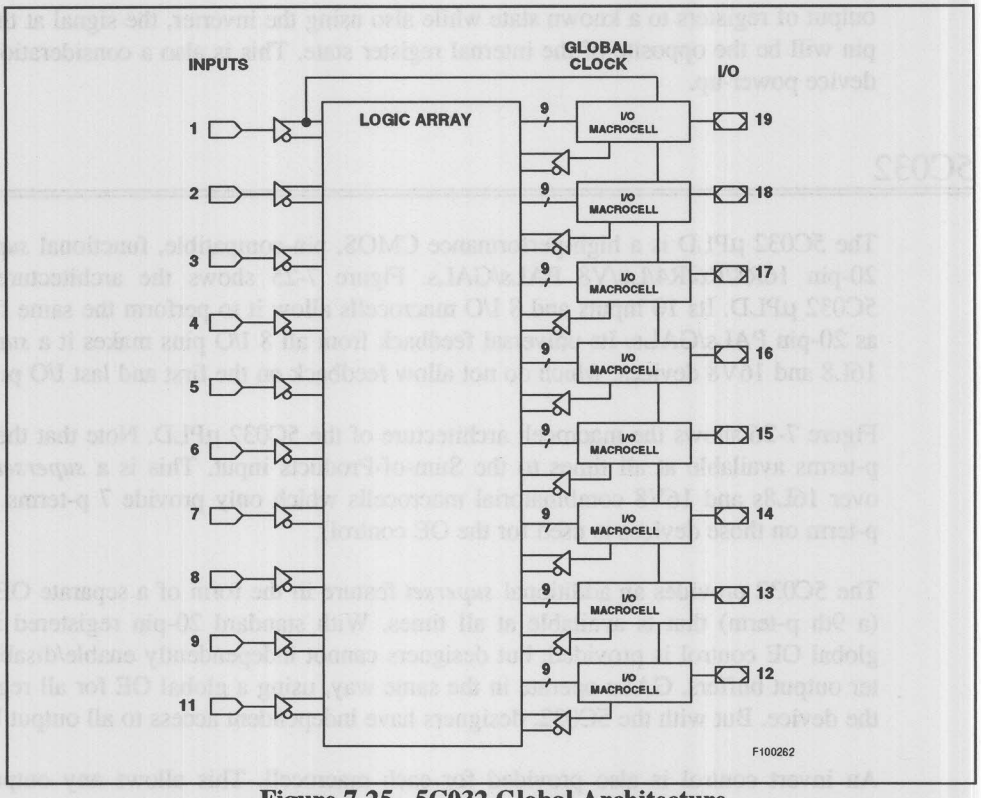


Figure 7-25. 5C032 Global Architecture

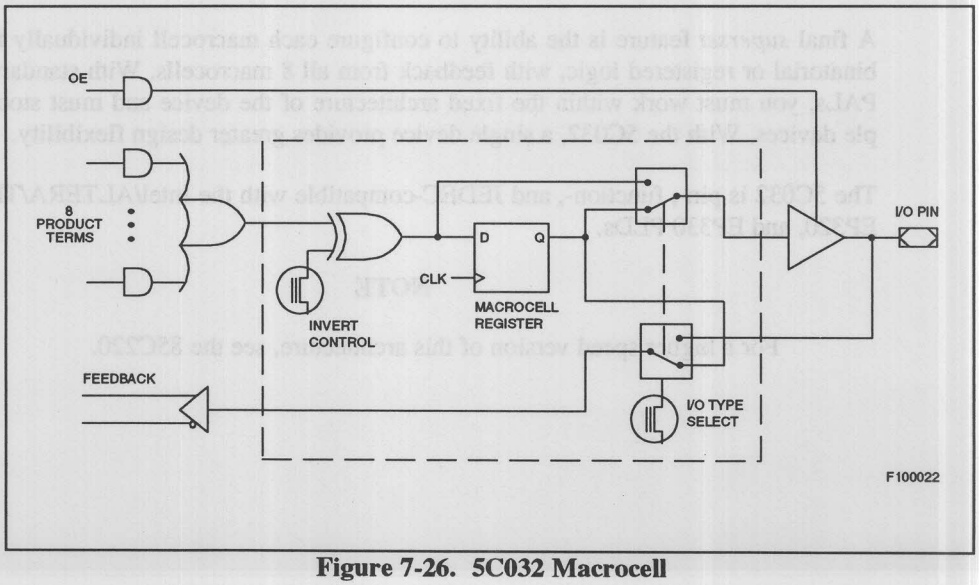


Figure 7-26. 5C032 Macrocell

5C060

The 5C060 is a standard architecture, high I/O count µPLD. As shown in Figure 7-27, the 5C060 features 16 I/O macrocells in a 24-pin package. Four dedicated inputs and two dedicated clocks complete the device architecture. CLK1 clocks the registers on half the device (8 registers); CLK2 clocks the registers on the other half (8 registers).

Figure 7-28 shows the macrocell architecture. Note that 8 p-terms are available to the SOP input at all times. Each macrocell contains an inversion control bit that allows any output to be individually configured as active high or active low. Finally, the OE p-term on each macrocell can be used as an asynchronous clock when not needed to control the macrocell output buffer. This allows up to 16 asynchronous clocks to be generated internally via logic equations.

The output from each macrocell can be independently configured as combinatorial or registered. Four different register types are available for each macrocell: D-type, T-type, JK-type, and SR-type. This wide range of features makes the 5C060 architecture ideally suited for register- and I/O-intensive designs. (JK- and SR-registers are emulations.)

Figure 7-29 shows the device clocking. There are two synchronous clock inputs. Each clock input drives eight registers. If more than eight registers are connected to the same synchronous clock input, both clock pins are needed to implement the circuit. In this event, the pin diagram on the Utilization Report shows both pins as clocks with a message to tie both pins together. Refer to Chapter 4, "PLDasm File/Language" for syntax information about clocking µPLDs.

NOTE

For a higher speed version of this architecture, see the iPLD610 and 85C060.

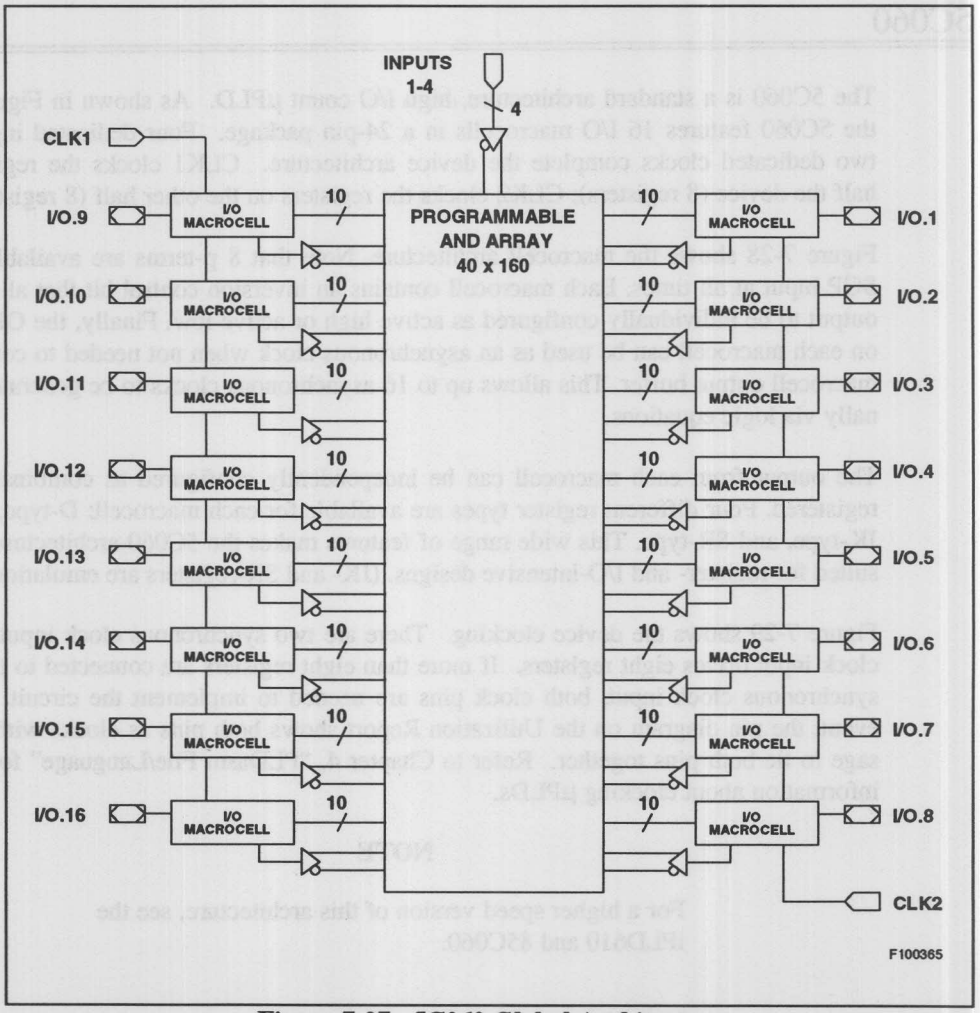


Figure 7-27. 5C060 Global Architecture

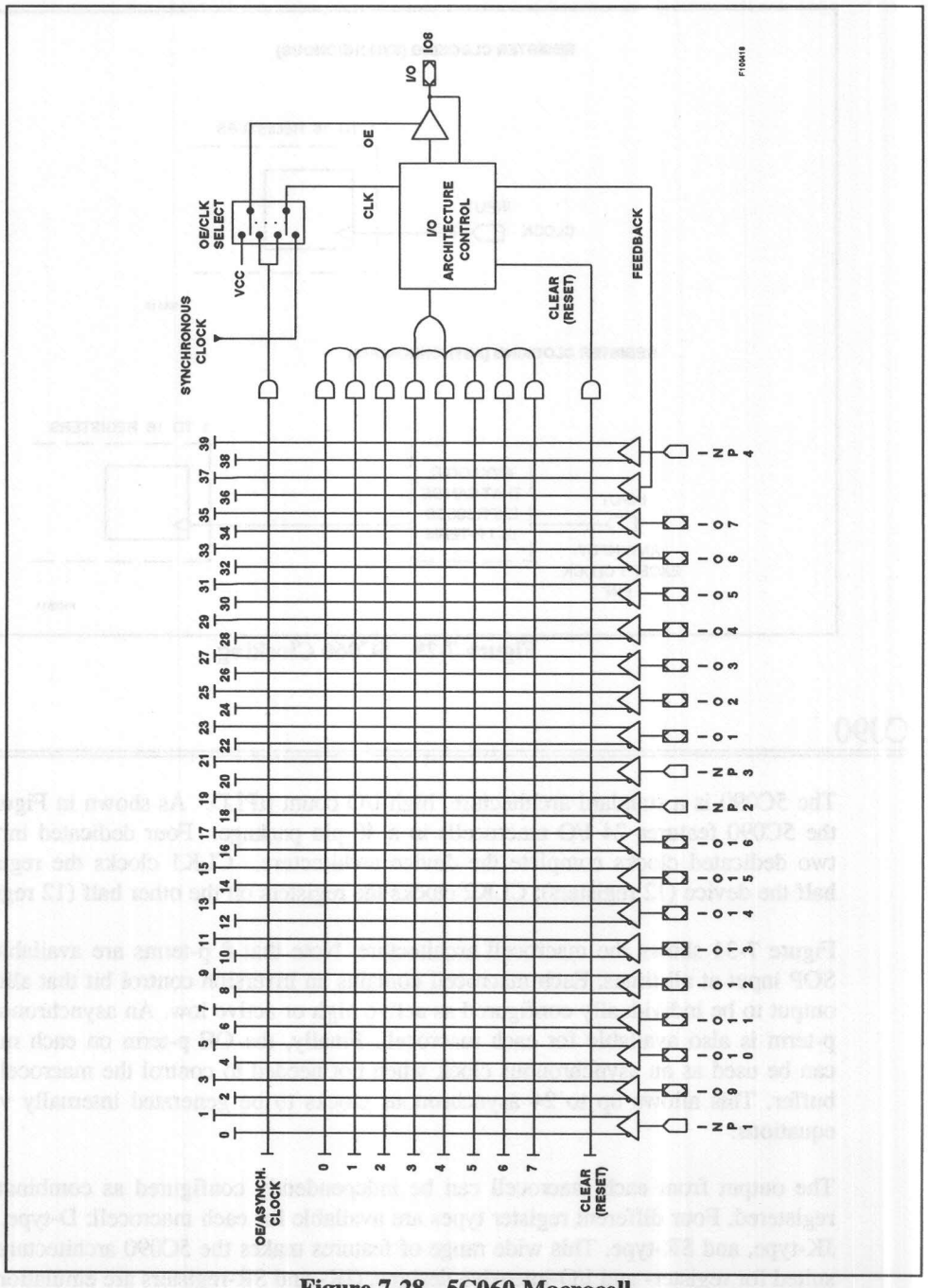


Figure 7-28. 5C060 Macrocell

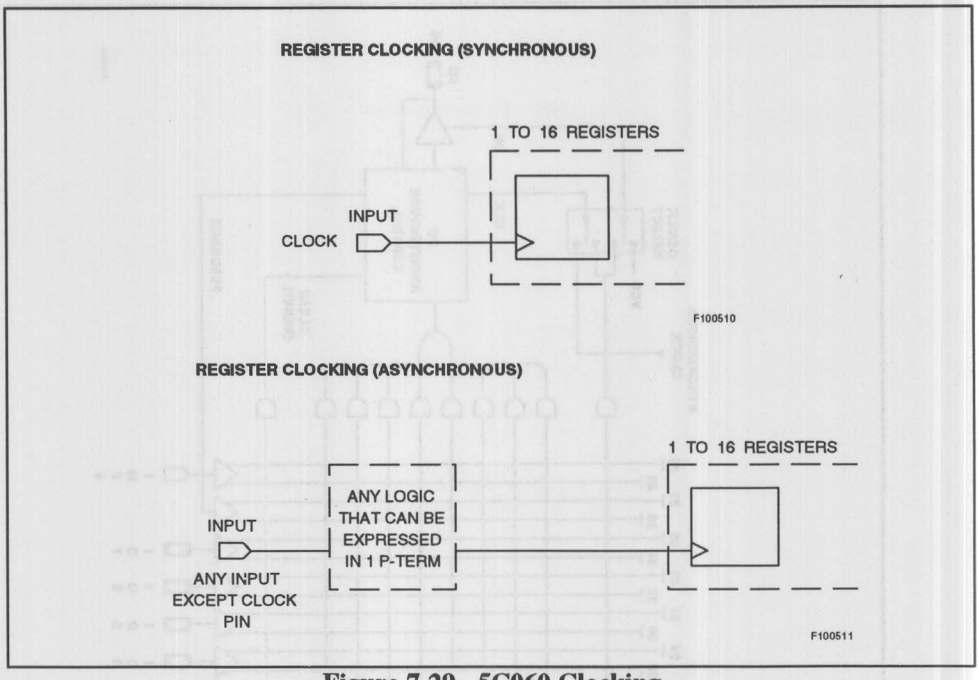


Figure 7-29. 5C060 Clocking

5C090

The 5C090 is a standard architecture, high I/O count µPLD. As shown in Figure 7-30, the 5C090 features 24 I/O macrocells in a 40-pin package. Four dedicated inputs and two dedicated clocks complete the device architecture. CLK1 clocks the registers on half the device (12 registers); CLK2 clocks the registers on the other half (12 registers).

Figure 7-31 shows the macrocell architecture. Note that 8 p-terms are available to the SOP input at all times. Each macrocell contains an inversion control bit that allows any output to be individually configured as active high or active low. An asynchronous clear p-term is also available for each macrocell. Finally, the OE p-term on each macrocell can be used as an asynchronous clock when not needed to control the macrocell output buffer. This allows up to 24 asynchronous clocks to be generated internally via logic equations.

The output from each macrocell can be independently configured as combinatorial or registered. Four different register types are available for each macrocell: D-type, T-type, JK-type, and SR-type. This wide range of features makes the 5C090 architecture ideally suited for register- and I/O-intensive designs. (JK- and SR-registers are emulations.)

Figure 7-32 shows the device clocking. There are two synchronous clock inputs. Each clock input drives 12 registers. If more than 12 registers are connected to the same synchronous clock input, both clock pins are needed to implement the circuit. In this

event, the pin diagram on the Utilization Report shows both pins as clocks with a message to tie both pins together. Refer to Chapter 4, "PLDasm File/Language" for syntax information about clocking µPLDs.

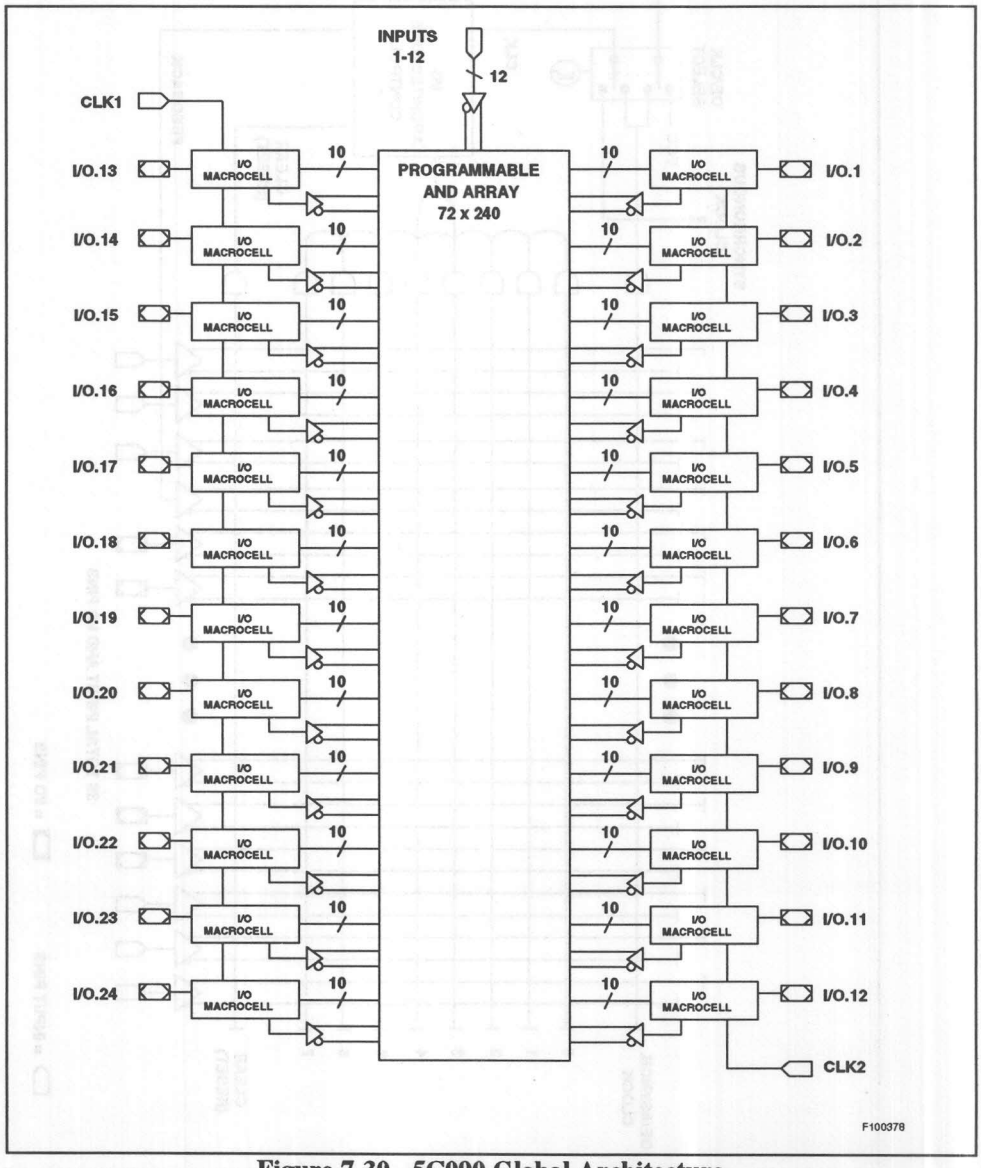


Figure 7-30. 5C090 Global Architecture

NOTE

For a higher speed version of this architecture, see the iPLD910 and 85C090.

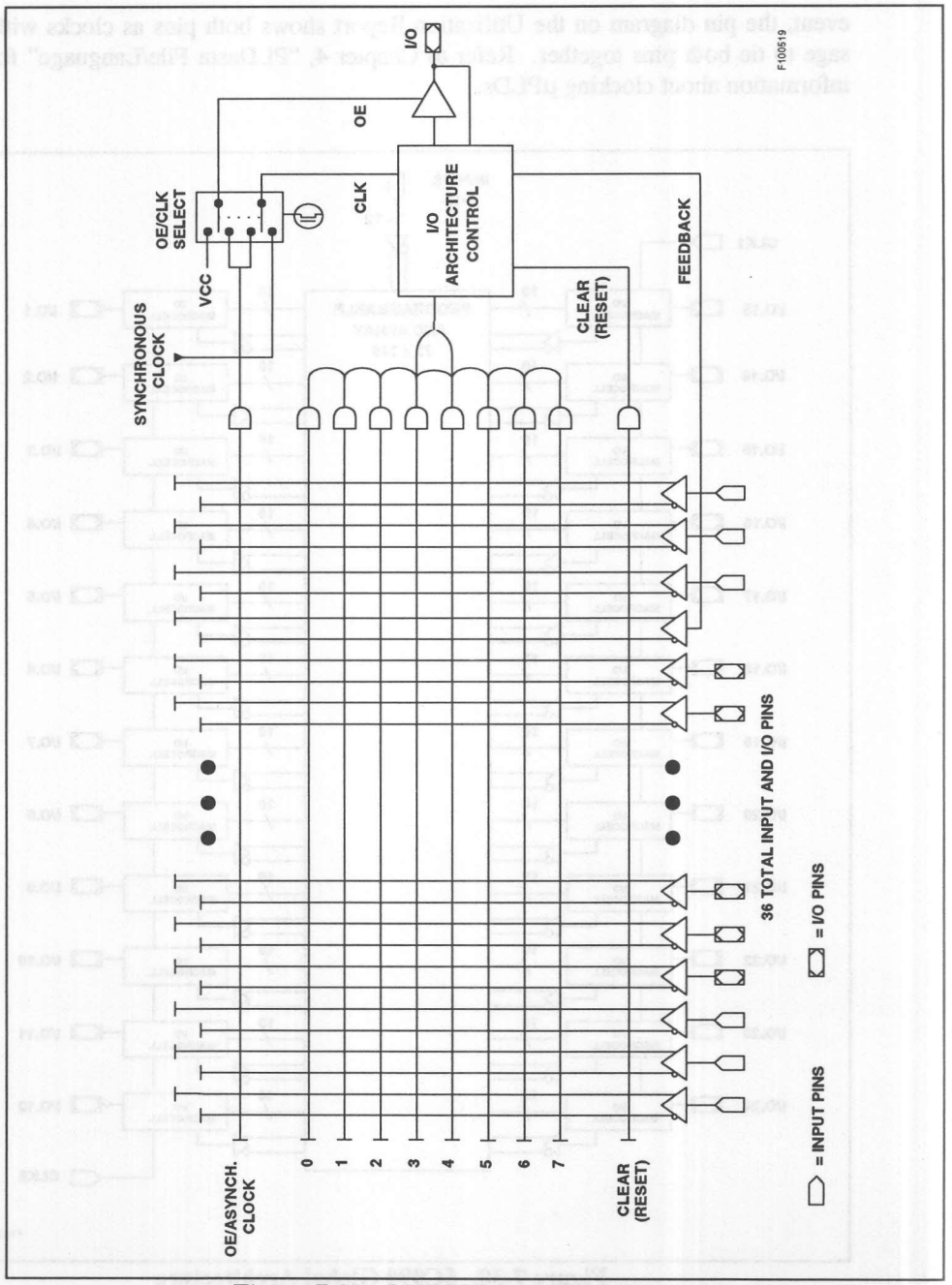


Figure 7-31. 5C090 Macrocell Architecture

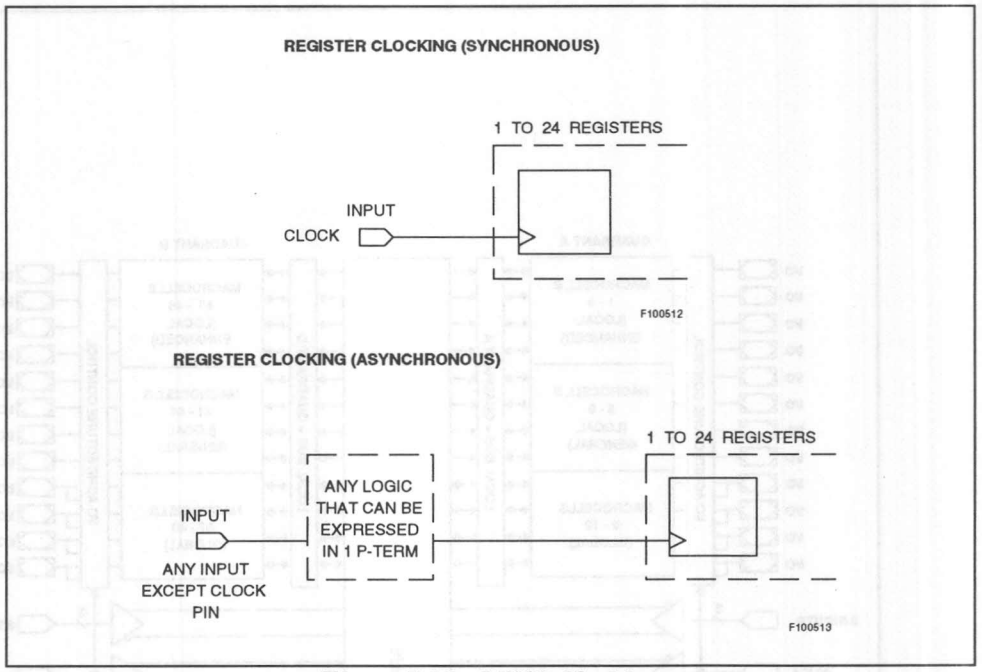


Figure 7-32. 5C090 Clocking

5C180

The 5C180 is a 68-pin µPLD that is divided into four identical quadrants A, B, C, and D. Each quadrant contains 4 dedicated inputs, one dedicated clock input and 12 programmable I/O pins. Each I/O pin is associated with a macrocell with 8 p-terms and an architecture control block (see Figure 7-33).

The architecture control block allows selection of D, T, JK, or SR registers or combinatorial output. JK- and SR-registers are emulations. Each quadrant is fed by a synchronous clock input. The clock inputs can also feed the p-term array. Each macrocell also supports a single p-term asynchronous clock input that can be used instead of the synchronous clock. There is also a single p-term clear and an output enable input. Table 7-7 shows the macrocell groups for the 5C180.

Within each quadrant, there are two different types of macrocells: Local Macrocells and Global Macrocells. Both types share an 88-input AND array and contain a total of 10 p-terms. Eight p-terms are dedicated for logic implementation. One p-term is reserved for Asynchronous Clear; one p-term is for Output Enable/Asynchronous Clock implementation.

Local Macrocells (see Figure 7-34) provide one feedback path into the AND array. Combinatorial, registered, or pin feedback can be selected from the Feedback Select

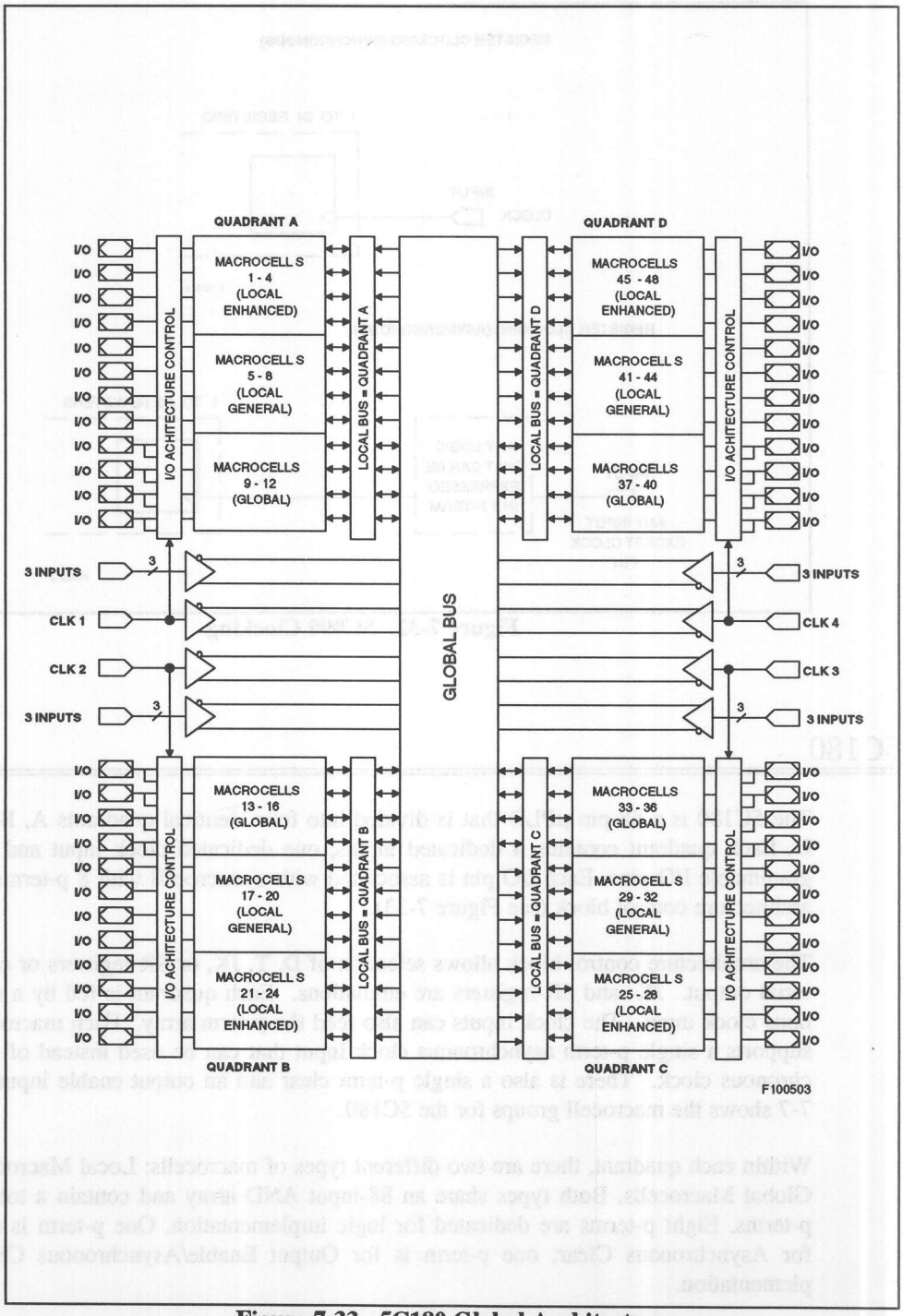


Figure 7-33. 5C180 Global Architecture

Table 7-7. 5C180 Macrocell Interconnections

	Pin Number	Macrocell Number	Feedback Structure	Feedback Inteconnect	Clock Group
Quad A	2-9 10-13	1-8 9-12	Local Local/Global	Quad A Quad A/All	1
Quad B	23-26 27-34	13-16 17-24	Local/Global Local	Quad B/All Quad B	2
Quad C	36-43 44-47	25-32 33-36	Local Local/Global	Quad C Quad C/All	3
Quad D	57-60 61-68	37-40 41-48	Local/Global Local	Quad D/All Quad D	4

Multiplexer. The selected feedback signal is then routed to the quadrant local bus. Therefore, the Local Macrocell feedback communicates only to macrocells within the same quadrant. There are a total of 32 Local Macrocells within the 5C180, with eight per quadrant.

Local Macrocells are subdivided into two groups: General Macrocells and Enhanced Macrocells. The Enhanced Macrocells are architecturally identical to the General Macrocells, but operate at higher speeds.

Global Macrocells (see Figure 7-35) contain two independent feedback paths to the AND array. Combinatorial or registered feedback is supplied to the local bus and pin feedback is supplied to the global bus. The "dual feedback" capability allows the macrocell to be used for internal logic functions as well as a dedicated input pin. To effectively use this configuration, the output buffer must be disabled. If the Global Macrocell I/O pin is not being used as a dedicated input, the macrocell logic can be fed back along the global bus allowing routing to any of the 5C180's 48 macrocells. There are 16 Global Macrocells contained in the 5C180, four per quadrant.

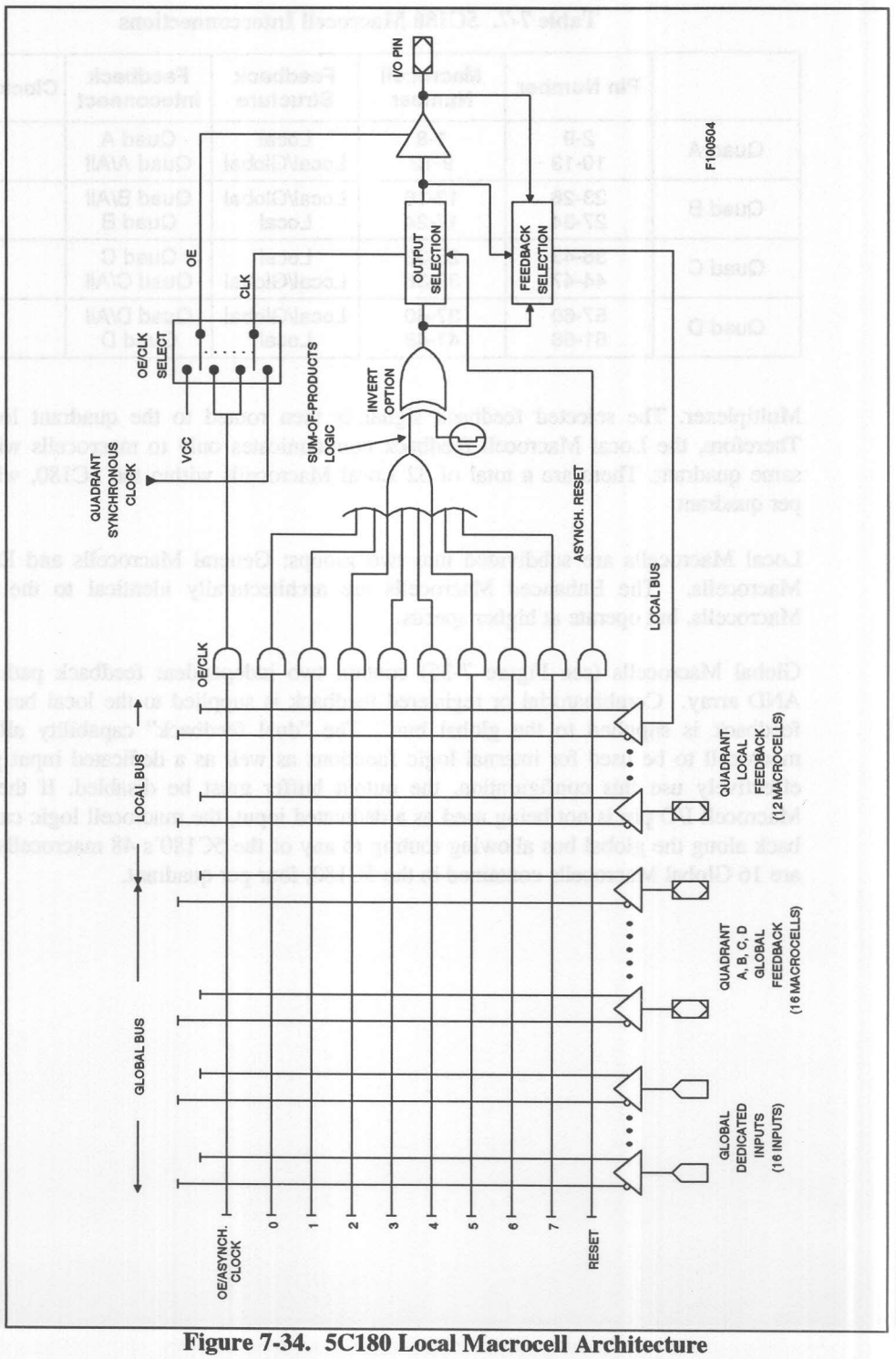


Figure 7-34. 5C180 Local Macrocell Architecture

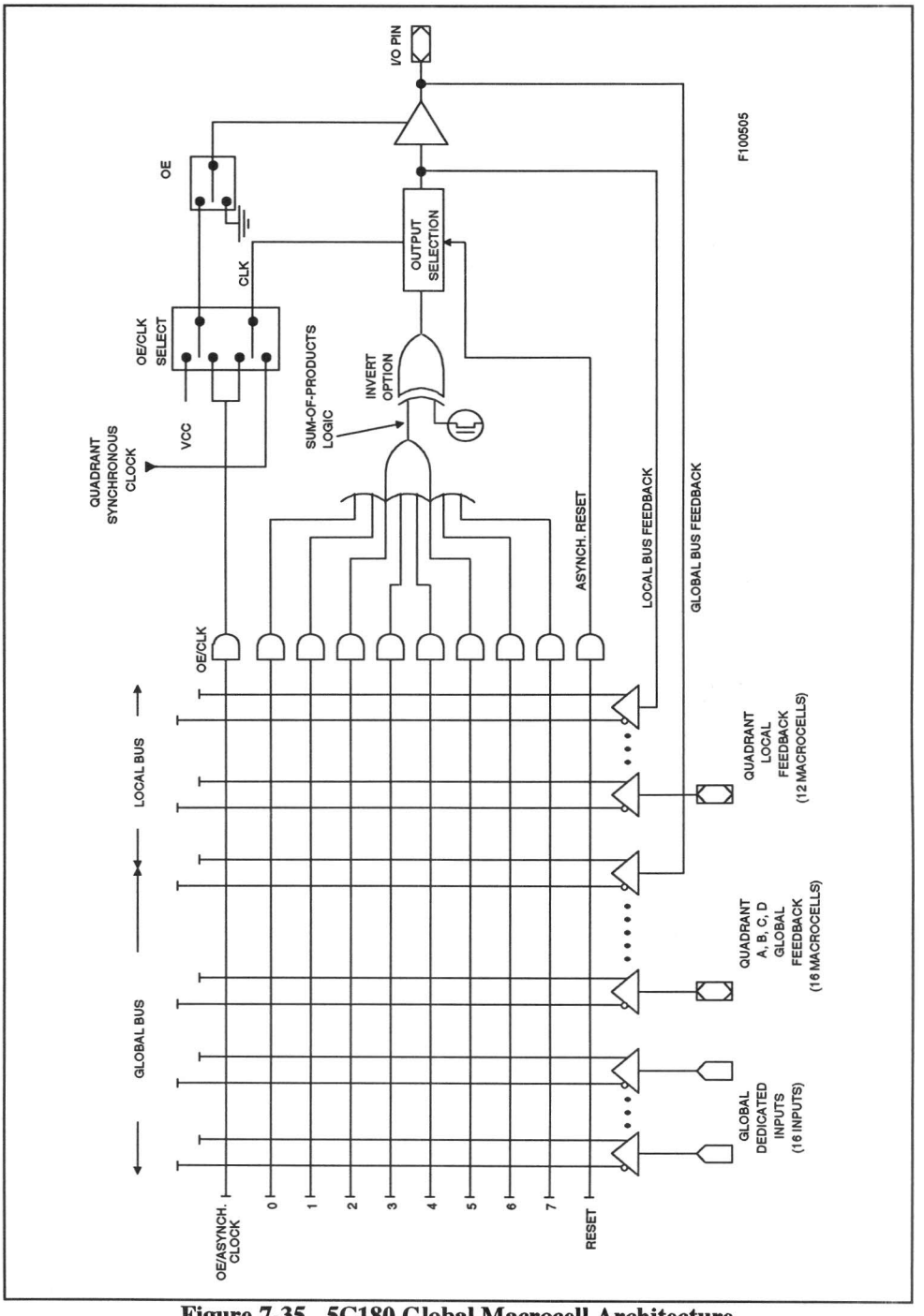
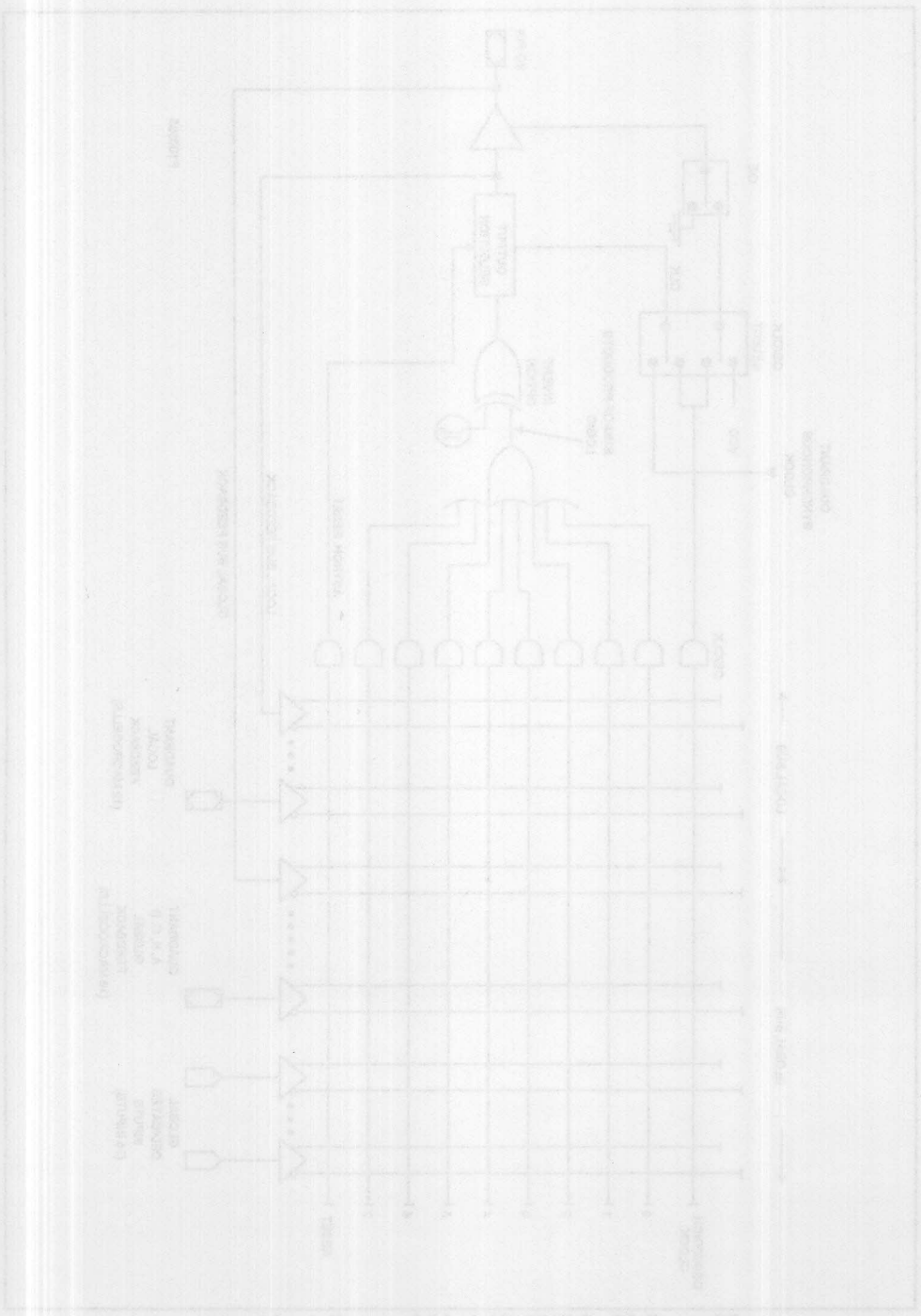


Figure 7-35. 5C180 Global Macrocell Architecture



Chapter 8 – Design Checklist

This chapter provides a checklist for designers using Intel µPLDs. This list is based on several years experience answering calls on our technical support Hot Line. The checklist is as follows:

1. Device Window Covered?
2. Unused Inputs and I/Os Grounded?
3. Turbo Bit On or Off?
4. Device Programmed Properly?
5. Adequate Decoupling on VCC and GND?
6. Good Ground Plane?
7. Inputs Active Before Power-up?
8. Operating Conditions Within Specification?
9. Good Connection at Leads?
10. Second Opinion?

Design Checklist

1. Device Window Covered?

Intel µPLDs are fabricated using Intel's standard EPROM process. Light entering through the window in CerDIP packages can disturb internal voltages and may result in indeterminate behavior. Covering the window with a label or UV filter ensures stable behavior.

2. Unused Inputs and I/Os Grounded?

Unused inputs on Intel's CMOS µPLDs are connected to the logic array, even when they are not used by the macrocells. When unused input pins are not tied high or low, they can cause devices to draw more power than necessary. To ensure the low power consumption, unused inputs should always be tied high or low.

For unused I/O pins, follow the directions for each pin as shown in the Report file (.RPT). You may be instructed to tie a pin to GND or to leave it alone (RESERVED).

3. Turbo Bit On or Off?

Most Intel µPLDs contain a Turbo Bit that optimizes device operation for speed or power savings. When the Turbo Bit is ON, the device will always be active and will always run at full speed. When the Turbo Bit is OFF, the device will enter standby mode if transitions are not detected on the input or I/O pins for a period of time. When powering up from standby mode in response to the next transition, an added delay is incurred.

Debugging problems can occur if a designer is expecting full-speed performance (assuming the Turbo Bit is ON) and the device responds more slowly (because the Turbo Bit is OFF). Designers should always know the state of the Turbo Bit for devices in their design.

When the Turbo Bit if OFF, but the device is being clock at higher frequencies, it will never enter standby mode. Refer to the "ICC vs. Frequency Graph" for each device in the *Programmable Logic* handbook to determine the frequency range where the device will enter standby mode.

4. Device Programmed Properly?

Debugging problems can occur if a µPLD is programmed with the wrong JEDEC file, or if the EPROM bits in a device are not fully programmed.

When experiencing problems with a design, check the following:

a. Verify the contents of the device against the original JEDEC file to make sure the correct design has been programmed into the device.

b. Program a second device using the same JEDEC file to see if it exhibits the same behavior.

c. Verify the device on a second programmer to catch any programmer problems.

5. Adequate Decoupling on VCC and GND?

Good decoupling practices should be followed for all designs using Intel µPLDs. This will also help minimize any system noise, especially in higher-speed designs. Like most other devices, Intel µPLDs can operate indeterminately when inputs are driven past the specified input threshold levels by noisy signals.

6. Good Ground Plane?

A ground plane should be used for all designs using Intel µPLDs. This will help minimize the impact of system noise on Intel µPLDs, especially in higher-speed designs. Like most other devices, Intel µPLDs can operate indeterminately when inputs are driven past the specified input threshold levels by noisy signals.

7. **Inputs Active Before Power-up?**

   Architecture bits in Intel µPLDs are loaded during device power-up. During power-up, devices are not guaranteed to respond to active inputs until after the specified power-up time (usually one microsecond). Refer to the power-up specification for each device data sheet in the *Programmable Logic* handbook for details.

8. **Operating Conditions Within Specification?**

   Intel µPLDs are specified to operate within certain voltage and temperature guidelines. If you are experiencing problems in a design, make sure that the system environment (voltage and temperature) does not exceed the published specifications for the devices.

9. **Good Connection at Leads?**

   Make sure that all leads on Intel µPLDs have good connections to the printed circuit board or socket. Check for bent pins and solder bridges at the device and socket junctions.

10. **Second Opinion?**

    Designers are sometimes so close to a design that they overlook a relatively simple logic problem. It often helps to have a second designer spend a few minutes looking over the design.

Chapter 9 – Sample Designs

Table 9-1 lists and summarizes sample designs that are useful in learning to design with PLDasm and Intel μPLDs. All examples are included with PLDshell Plus in your installation directory. Some of these files are used as illustrations in this manual. The AP-xx and AB-xx references indicate Application Notes/Briefs in the *Programmable Logic* handbook.

Table 9-1. Example Files

Filename	Description
File/Language Examples	
TEMPLATE.PDS	Template file containing blank fields and major keywords. Can be used to get a design started. This is a reference file only, not a working design.
SUMMARY.PDS	Summary file showing examples of main PLDasm syntax elements in a meaningful context. This is a quick reference file only, not a working design.
Boolean Equations	
4COUNT.PDS	4-bit synchronous counter in Boolean equations. Target device is an 85C224. Illustrates basic registered circuit design using Boolean equations.
4ERROR.PDS	Same design as 4COUNT.PDS, but with an intentional error in the QA equation. Target device is an 85C224. Used to illustrate the on-line error help feature. See "Getting Started".
PS2POS.PDS	Programmable Option Select for PS/2 Adaptor card in Boolean equations. Target device is a 5AC312. Design is described in AP-317.
CASCADE.PDS	Large XOR function in Boolean equations. Target device is an 85C220. Illustrates how to distribute large equations across macrocells to help fit designs. Circuit is described in AB-8.
16COUNT.PDS	16-bit binary counter in Boolean Equations. Target device is an 85C060. Circuit is described in AB-11.
24COUNT.PDS	24-bit binary counter in Boolean Equations. Target device is an 85C090. Illustrates use of Toggle Flip-Flops to simplify counter design. Based on original 16-bit counter described in AB-11.
Truth Tables	
7SEG.PDS	7-segment decoder in Truth Table format. Target device is an 85C220. Slightly modified from original file shipped with PLDshell. Illustrates Truth Table use for a simple combinatorial design.
ADDR1.PDS	Address decoder in Truth Table format. Target device is an 85C508. Circuit is described in AP-322.
TCOUNT.PDS	Counter implemented in Truth Table format and converted to T-type Flip-Flops. Target device is an 85C060. Illustrates how to use Truth Tables to implement counters.

Table 9-1. Example Filenames (Continued)

Filename	Description
State Machine Designs	
2BIT.PDS	2-bit Up/Down Counter in State Machine format. Target device is an 85C220. Illustrates use of State Machine syntax in a simple design.
BUSCON1.PDS	Simple Bus Controller circuit in State Machine format. Target device is an 85C224. Illustrates State Machine format for a simple design.
DOUBLCNT.PDS	Two 4-bit counters, one synchronous, one asynchronous in State Machine format. Target device is an 85C060. Illustrates use of State Machine syntax in a more complex design. Also illustrates asynchronous clocking of state machines.
EXMEALY1.PDS	Mealy State Machine example in State Machine format. Target device is an 85C224. Used to illustrate Mealy outputs in a state machine.
Application Examples (Mixed Format)	
UPDOWN.PDS	Up/down counter in State Machine format and Boolean equations. Device-independent design. Can be fitted to a variety of devices. Used to illustrate the differences between device-indpendent and device-specific deisgn.
STATEDEC.PDS	State machine feeding a decoder in State Machine format and Boolean equations. Target device is an 85C060.
MEMCONT.PDS	Shared memory controller for EISA add-in card in State Machine format and Boolean equations. Target device is an 85C060. Based on original circuit described in AP-339 (some changes have been made).
PULSE1.PDS	Pulse generator examples in State Machine format and Boolean equations. Target device is an 85C060.
Disassembly/Conversion Examples	
EX16R6.JED	JEDEC file for a 16R6 design. Used to illustrate JEDEC disassembly/conversion to Intel µPLDs. Target device is an 85C220.
EX20L8.JED	JEDEC file for a 20L8 design. Used to illustrate JEDEC disassembly/conversion to Intel µPLDs. Target device is an 85C224.
EX20V8.JED	JEDEC file for a 20V8 design. Used to illustrate JEDEC disassembly/conversion to Intel µPLDs. Target device is an 85C224.

Table 9-1. Example Filenames (Continued)

Filename	Description
ADF/SMF Translation Examples	
SAMP1.ADF	Sample 4-bit store and increment circuit written in ADF format. Target device is a 5AC312. Used to illustrate the ADF-to-PDS translation utility.
MANYMACH.SMF	Muliple state machine file in SMF format. Target device is an 5AC312. Used to illustrate the SMF-to-PDS translation utility.
MACFILE.SMF	State machine and TTL macro circuit in SMF format. Target device is a 5C060. Used to illustrate the SMF-to-PDS translation utility.

Appendix A – Language Reference Summary

This appendix lists PLDasm information for quick reference. See also the file TEMPLATE.PDS in your installation directory.

Keywords and Reserved Words

The following words have defined meaning within PLDasm source files and cannot be used as pin names or signal names. Keywords and reserved words are listed in alphabetical order.

BEGIN
BURIED
CHECK
CHIP
CLOCKF
CMBFBK
COMB
COMBINATORIAL
CONDITIONS
DEFAULT_BRANCH
DEFAULT_OUTPUTS
DO
ELSE
END
EQUATIONS
FOR
GND
HIGH
HOLD_STATE
IF
INPUT
I/O
LAT
LATCHED
LATFBK
LOW
MEALY_MACHINE
MOORE_MACHINE
NC
NEXT_STATE
OFF
ON
OPTIONS
OR
OUTPUT
OUTPUT_HOLD
PIN
PINFBK
PRELOAD
PRLDF
REG
REGFBK
REGISTERED
SETF
SIMULATION
SIGNATURE
STATE
STRING
THEN
TO
TRACE_OFF
TRACE_ON
T_TAB
TURBO
VCC
VECTOR
WHILE

Boolean Operators

Operator	Description
/	Active-Low in pin declaration/Boolean NOT elsewhere
*	Boolean AND
+	Boolean OR
:+:	Boolean XOR
=	Combinatorial Output
*=	Latched Output
:=	Registered Output

Signal Extensions

Extension	Description
.D	Data Input to D-type Register
.J	J Data Input to JK Register (emulation)
.K	K Data Input to JK Register (emulation)
.R	R Data Input to SR Register (emulation)
.S	S Data Input to SR Register (emulation)
.T	Data Input to Toggle Register
.ACLK	Asynchronous Clock (p-term)
.ALE	Asynchronous Latch Enable (p-term)
.LE	Synchronous Latch Enable
.CLKF	Clock Pin (Synch.) or Clock Equation (Asynch.)
.TRST	Output Enable Equation
.RSTF	Clear Equation
.SETF	Preset Equation
.OUTF	State Machine Output
.FB	Feedback (sometimes useful for simulation)

=	Equals
>	Greater than
<	Less than
>=	Greater than or equal to
<=	Less than or equal to
/=	Does not equal

The following facing pages show the file SUMMARY.PDS, which illustrates many of these language elements in a meaningful context. This is a reference file only, not a working design.

```
Title           PLDasm Language Summary
Pattern         <pattern label>
Revision        <rev. number>
Author          <your name>
Company         <your company>
Date            <current date>

OPTIONS
     TURBO = [ON|OFF]          ; default is ON
     SECURITY = [ON|OFF]       ; default is OFF

;          design name    partname
CHIP       template       85Cxxx
;
; Some Available Parts:
;          partname  speed bins
;    20 pin    C/PDIP    85C220       -80(10ns Tpd),-66(12ns Tpd)
;              PLCC      N85C220      -7, -80(10ns Tpd), -66(12ns Tpd)
;              C/PDIP    5C032        -30, -35,-40
;
;    24 pin    C/PDIP    iPLD22V10    -10, -15
;              PLCC      iPLD22V10N   -10, -15
;              C/PDIP    iPLD610      -10, -15, -25
;              PLCC      iPLD610N     -10, -15, -25
;              C/PDIP    85C224       -80(10ns Tpd), -66(12ns Tpd)
;              PLCC      N85C224      -7, -80(10ns Tpd), -66(12ns Tpd)
;              C/PDIP    85C060       -10, -12, -15
;              PLCC      N85C060      -10, -12, -15
;              C/PDIP    5AC312       -25, -30
;              PLCC      N5AC312      -25, -30
;              C/PDIP    85C22V10     -10, -15
;              PLCC      N85C22V10    -10, -15
;              C/PDIP    5C060        -45, -55
;              PLCC      N5C060       -45, -55
;
;    28 pin    C/PDIP    85C508       -7, -10     ; Decoder/Latch PLD
;              PLCC      N85C508      -7, -10
;
;    40 pin    C/PDIP    iPLD910      -12, -15, -20, -25
;              PLCC      iPLD910N     -12, -15, -20, -25
;              C/PDIP    85C090       -12, -15, -20, -25
;              PLCC      N85C090      -12, -15, -20, -25
;              C/PDIP    5AC324       -25, -30, -35
;              PLCC      N5AC324      -25, -30, -35
;              C/PDIP    5C090        -50, -60
;              PLCC      N5C090       -50, -60
;
;    68 pin    PLCC      N5C180       -70, -75, -90
;              PGA       A5C180       -70, -75, -90
;
; For futher partname information, see Table 1-1 in the PLDshell
; Plus Manual. Check device data sheets for speed/timing
; information. Extended Temperature/Military versions of most
; devices are available.
```

```
; PINLIST

PIN  1    CLOCK

PIN       UPDOWN     ; undefined pin assignment
PIN       CLEAR

PIN  2    I1         ; Direct Inputs
PIN  3    I2
PIN  4    I3
PIN  5    I4

PIN       Q0         ; unassigned state variables a state machine
PIN       Q1
PIN       Q2
PIN       Q3

; - Output Types -
     ; Combinatorial Output, I/O Feedback
     PIN            O1         COMB, PINFBK
     ; Combinatorial Output, MC Feedback
     PIN            O2         COMB, CMBFBK
     ; Combinatorial Output, Reg Feedback (85C22V10 only)
     PIN            O3         COMB, REGFBK
     ; Buried Combinatorial Macrocell
     PIN            O4         CMBFBK, BURIED
     ; Registered Output, I/O Feedback
     PIN            O5         REG, PINFBK
     ; Registered Output, MC Feedback
     PIN            O6         REG, REGFBK
     ; Buried Register
     PIN            O7         REGFBK, BURIED

; - Input Types -
     ; Latched Input      (5AC312/5AC324)
     PIN            5          LIN1  LATCHED
     ; Registered Input   (5AC312/5AC324)
     PIN            6          RIN1  REG

; - String Substitutions throughout file -
     STRING    QADS       ' (ADS * PCLK * /RESET) '
     STRING    RASON      ' ((RAS0 + RAS1) * /IREADY) '

;  Design Sections can appear in any order; the Simulation
;  section must appear last
;
;    [ STATE ]            - can have multiple machines
;    [ EQUATIONS ]        - only one equation section allowed
;    [ T_TAB ]            - can have multiple truth tables
;    [ SIMULATION ]       - only one simulation section allowed
;
;    At least one STATE/EQUATIONS/T_TAB section must appear.
```

```
; - State Machine Format -
STATE   [MEALY_MACHINE|MOORE_MACHINE]

;                   1    0    specifies output values on
;                             hold conditions
     OUTPUT_HOLD    OUT1 /OUT2
;                   0         1    X         specifies default output
;                                              values
     DEFAULT_OUTPUT /OUT1     OUT2 %OUT3
;                                   branches for unresolved states
     DEFAULT_BRANCH S0             ; go to S0
     DEFAULT_BRANCH HOLD_STATE     ; stay in current state
     DEFAULT_BRANCH NEXT_STATE     ; go to next state in assignments
                                   ; list

; State assignments, value of the machine variables for each state.
; Gray code state assignments for a two-bit machine, S0-S3

     S0 = /Q1 * /Q0      ; power-up state of Intel PLD Registers
     S1 = /Q1 *  Q0
     S2 =  Q1 *  Q0
     S3 =  Q1 * /Q0

; Gray code state assignments for a three-bit machine, S0-S7

     S0 = /Q2 * /Q1 * /Q0
     S1 = /Q2 * /Q1 *  Q0
     S2 = /Q2 *  Q1 *  Q0
     S3 = /Q2 *  Q1 * /Q0
     S4 =  Q2 *  Q1 * /Q0
     S5 =  Q2 *  Q1 *  Q0
     S6 =  Q2 * /Q1 *  Q0
     S7 =  Q2 * /Q1 * /Q0

; Gray code state assignments for a four-bit machine, S0-SF

     S0 = /Q3 * /Q2 * /Q1 * /Q0  ; 0x0
     S1 = /Q3 * /Q2 * /Q1 *  Q0  ; 0x1
     S2 = /Q3 * /Q2 *  Q1 *  Q0  ; 0x3
     S3 = /Q3 * /Q2 *  Q1 * /Q0  ; 0x2
     S4 = /Q3 *  Q2 *  Q1 * /Q0  ; 0x6
     S5 = /Q3 *  Q2 *  Q1 *  Q0  ; 0x7
     S6 = /Q3 *  Q2 * /Q1 *  Q0  ; 0x5
     S7 = /Q3 *  Q2 * /Q1 * /Q0  ; 0x4
     S8 =  Q3 *  Q2 * /Q1 * /Q0  ; 0xC
     S9 =  Q3 *  Q2 * /Q1 *  Q0  ; 0xD
     SA =  Q3 *  Q2 *  Q1 *  Q0  ; 0xF
     SB =  Q3 *  Q2 *  Q1 * /Q0  ; 0xE
     SC =  Q3 * /Q2 *  Q1 * /Q0  ; 0xA
     SD =  Q3 * /Q2 *  Q1 *  Q0  ; 0xB
     SE =  Q3 * /Q2 * /Q1 *  Q0  ; 0x9
     SF =  Q3 * /Q2 * /Q1 * /Q0  ; 0x8

; state transitions

     S0 := VCC       -> S1; on next clock go to S1
     S1 := UP        -> S2
        + DOWN       -> S4
```

```
; output transitions, MOORE
; Moore outputs are default transitions (VCC) only

     S1.OUTF := VCC      -> LOCAL * /MEMORY * /INTACK ; registered
     S2.OUTF  =  VCC     -> ASTRB                     ; combinatorial
     S3.OUTF  =  VCC     -> ASTRB

; output transitions, MEALY
; Mealy's may have conditions on the output transitions

     S1.OUTF := DOWN     ->  LOCAL * /MEMORY * /INTACK ; registered
              + UP       -> /LOCAL
     S2.OUTF =  UP       -> ASTRB       ; combinatorial
     S3.OUTF  =  VCC      - ASTRB


; input conditions that determine state and output transitions
     CONDITIONS
          UP    =  UPDOWN * /CLEAR
          DOWN = /UPDOWN * /CLEAR
          ACTIVE = /EN + RDY

; Moore machines are level sensitive. The outputs do not change
; until the next clock edge.  Mealy machines can generate pulse
; signals. The outputs may change before the next clock edge.
; See also the examples in exmealy1.pds and pulse1.pds.

; - Boolean Equations section -
EQUATIONS

     O1 = ...        ; - Combinatorial Output     (COMBINATORIAL)
     O1 := ...       ; - Registered (D) Output    (REGISTERED)
        O1.FB        ; - Feedback path from macrocell (REGFBK, CMBFBK)
     O1.IO           ; - Feedback path from I/O pin    (PINFBK)
     O1.D := ...     ; - Registered (D) Output    (REGISTERED)
     O1.T := ...     ; - Toggle (T) Output
     O1.J := ...     ; - J/K Output (emulated J/K.
                     ;    Synchronized with .CLKF)
     O1.K := ...
     O1.S := ...     ; - S/R Output (emulated J/K.
                     ;    Synchronized with .CLKF)
     O1.R := ...

; - Control Signals -
     O1.CLKF = CLOCK                ; Register clock signal
     O1.ACLK = CLOCK * ENABLE       ; Asynchronous clock signal,
                                    ; from pterm array
     ; See Chapter 7 in the PLDshell Plus Manual for specific
     ; device clocking options, and macrocell control signals.
     O1.RSTF = CLEAR                ; Register Clear signal
     O1.SETF = PRESET               ; Register Preset signal
     O1.TRST = /OE                  ; OE signal

; - Logic -
      AND1= IN1 * IN2               ; Logical AND
     /NAND1     = IN1 * IN2         ; Logical NAND
      OR1       = IN1 + IN2         ; Logical OR
     /NOR1      = IN1 + IN2         ; Logical NOR
      XOR1= IN1 :+: IN2             ; Logical XOR
     /XOR1      = IN1 :+: IN2       ; Logical XNOR
      NOT       = /IN1              ; Logical NOT
```

```
; - Truth Table section -
T_TAB    ; combinatorial truth table
     ( I1 I2 I3 I4  >> C1 C2 C3 C4 )
       1  0  0  0  :  1  0  0  0
       0  1  0  0  :  0  1  0  0
       0  0  1  0  :  0  0  1  0
       0  0  0  1  :  0  0  0  1

T_TAB    ; registered truth table
     ( I1 I2 I3 I4  :> R1 R2 R3 R4 )
       1  0  0  0  :   1  0  0  0
       0  1  0  0  :   0  1  0  0
       0  0  1  0  :   0  0  1  0
       0  0  0  1  :   0  0  0  1


; - Simulation section -
SIMULATION

     ; - Build vectors of outputs to use as tests for
     ;IF's and WHILE's

     VECTOR INS      := [ IN8,IN7,IN6,IN5,IN4,IN3,IN2,IN1,IN0 ]
     VECTOR NUM      := [ Q3, Q2, Q1, Q0 ]
     VECTOR GLOB     := [ ADDR23, ADDR22, ADDR16, ADDR15, ADDR12 ]

     ; - Set all inputs to known values
     ;      0        0    1   0   1
     SETF /CLKPIN /ILE  I1 /I2 /I3  INS:=0377

     ; - Preload registers to known a state (Intel PLD registers
     ;    power-up to 0) NOTE: See PLDshell Plus Manual about preload
     ;    for specific PLD's.

     PRLDF /Q0 /Q1 /Q2 /Q3

     ; - Clock an input signal  0 -> 1 -> 0

     CLOCKF CLKPIN

     ; - CHECK output values, report any mismatches
     ;      1   0   0   0   1   1
     CHECK O1 /O2 /O3 /O4  O5  O6

     ; - FOR loop to count up 6 clocks

     FOR j := 0 TO 5 DO
     BEGIN
          SETF INS := j
          CLOCKF CLK
          IF ( NUM == 4 )      ; when in state 4
          BEGIN
               SETF /OE        ; disable OE
          END
```

Appendix B – Utilization Report

The Utilization Report shows which of a device's resources have been used by a particular design and how they have been used. If selected as a compile option, the report is automatically stored and listed in the file directory as <filename>.RPT.

NOTE

If you specify all pin assignments in the design and the design fits, there is no need to refer to this report. However, if some pin assignments are unspecified or if there are fitting problems, refer to this report to determine which pin assignments were selected by the Fitter or to isolate fitting problems.

Utilization Report Sections

A Utilization Report contains: a header, a listing of the compiled source code, up to four tables of fitting information, and device utilization statistics. The tables include:

- INPUTS
- OUTPUTS
- UNUSED RESOURCES
- MACROCELL INTERCONNECTION CROSS REFERENCE

Header and Source Listing

The header information is the same as in the compiler source file as shown in Figure B-1. The source code that was used by the compiler to produce the Utilization Report follows the header information. Note that comments are not included and that all design sections have been reduced to Boolean equation form. If minimization was selected during compilation, equations are in their minimized form.

```
INTEL Logic Optimizing Compiler Utilization Report
FIT Release [ Vx.y ] SID [ x.yyy ]


***** Design implemented successfully

Title                4-Bit Counter Sample File
Pattern              pds
Revision             1
Author               Your Name
Company              Your Company
Date                 Date

(SOURCE CODE OF COMPILED DESIGN)
```

Figure B-1. Report Header

Pin Connections

When a design is successfully implemented, the Utilization report contains a diagram of the device pinouts with signal names (see Figure B-2). Notes about pins may be present below the pinout diagram.

Inputs Table

The Inputs table indicates where each input is fitted and which resource was used. Figure B-3 shows an Inputs table for a simple 4-bit counter.

If the input is fitted on an I/O pin, the Macrocell number (MCell #) is given and the PTerms column contains a zero.

The resources column refers to the type of input. In the example, INP means Pin Input to Logic Array.

```
***** Design implemented successfully

              85C224
            - - - - -
        CLK -| 1   24|- Vcc
        ENA -| 2   23|- Gnd
        Gnd -| 3   22|- Gnd
        Gnd -| 4   21|- Gnd
        Gnd -| 5   20|- Gnd
        Gnd -| 6   19|- Gnd
        Gnd -| 7   18|- QD
        Gnd -| 8   17|- QC
        Gnd -| 9   16|- QB
        Gnd -|10   15|- QA
        Gnd -|11   14|- Gnd
        GND -|12   13|- Gnd
            - - - - -

CMOS Device: ground unused inputs and I/Os.
Gnd = unused input or I/O pin.
RESERVED = Leave pins unconnected on board.
N.C. = unconnected pin.
```

Figure B-2. Pinout Diagram

```
**INPUTS**
      Name   Pin   Resource   MCell   PTerms
       ENA     2        INP       -       -
       CLK     1        INP       -       -
```

Figure B-3. Inputs Table

Outputs Table

The Outputs table shows where each output is fitted (see Figure B-4).

The Resource entry gives the name of the I/O primitive for the output signal. In the example, RORF refers to D-Register pin Output, Register Feedback.

The MCell entry tells which macrocell the output is connected to.

The PTerm entry tells the number of product-terms used in the equation, and the number available. For JK and SR flip-flop primitives, this is the total number of product-terms for both equations.

```
**OUTPUTS**
      Name   Pin   Resource   MCell   PTerms
        QA    15       RORF       8     1/ 8
        QB    16       RORF       7     2/ 8
        QC    17       RORF       6     3/ 8
        QD    18       RORF       5     4/ 8
```

Figure B-4. Outputs Table

Unused Resources

When a design does not use all the resources of a device, the Unused Resources Table is generated. Figure B-5 shows the Unused Resources table for the 4-bit counter.

```
**UNUSED RESOURCES**
      Name   Pin   Resource   MCell   PTerms
         -     3      INPUT       -        -
         -     4      INPUT       -        -
         -     5      INPUT       -        -
         -     6      INPUT       -        -
         -     7      INPUT       -        -
         -     8      INPUT       -        -
         -     9      INPUT       -        -
         -    10      INPUT       -        -
         -    11      INPUT       -        -
         -    13      INPUT       -        -
         -    14      INPUT       -        -
         -    19      MCELL       4        8
         -    20      MCELL       3        8
         -    21      MCELL       2        8
         -    22      MCELL       1        8
         -    23      INPUT       -        -
```

Figure B-5. Unused Resources Table

Part Utilization

The Part Utilization Table shows the number and percentage of pins and macrocells, and the percentage of product-terms fitted into the target device (see Figure B-6).

```
**PART UTILIZATION**
 4/ 8  MacroCells (50%), 31% of used Pterms Filled
 2/14  Input Pins (14%)
       PTerms Used 15%
```

Figure B-6. Part Utilization

Macrocell Interconnection Cross Reference

This table shows each macrocell feedback or input interconnection. Figure B-7 shows the macrocell interconnection table for the 4-bit counter example.

Feedbacks and inputs to the logic array are listed on the left side. Feedbacks show the primitive used and the macrocell number preceded by an @ symbol. Inputs show the pin number preceded by an @ symbol. Destination macrocells are listed on top with their corresponding signal name on the bottom. The column on the right side shows the pins associated with macrocells using pins used for outputs. Output pins are preceded by an @ symbol.

An "x" at a column intersection indicates that no feedback signal is possible due to device architecture (i.e., the feedback is a local feedback only). A period (.) at a column intersection indicates that a connection is possible, but none has been implemented. An asterisk (*) at a column intersection indicates that a feedback or input connection exists. A question mark (?) indicates that a feedback or input connection was attempted where none is possible on the target device.

```
Macrocell Interconnection Cross Reference          4COUNT.rpt
FEEDBACKS:                  M M M M
                            0 0 0 0
                            5 6 7 8
QD ....... RORF @M5 -       * . . . @18
QC ....... RORF @M6 -       * * . . @17
QB ....... RORF @M7 -       * * * . @16
QA ....... RORF @M8 -       * * * * @15
INPUTS:
CLK ...... INP  @1  -       * * * *
ENA ...... INP  @2  -       * * * *
                            Q Q Q Q
                            D C B A
. = not connected             x = no connection possible
* = signal feeds cell        ? = error, unable to fit
```

Figure B-7. Part Utilization

Appendix C – PLDshell Configuration File

This section describes the PLDSHELL.CFG configuration file used by PLDshell Plus/PLDasm. The configuration file is distributed with PLDshell Plus, and is copied to the install directory during installation. Figure C-1 shows the full format of the configuration file. If the defaults are used, many entries will not be present until changes are made.

The following is a list of the supported variable assignments and a description of their functions:

IPLSPATH	Specifies directory path(s) to the PLDshell Plus and associated files, which are usually in the same directory.
INCLUDE	Specifies directory path to DPP macro files.
PLD_MSG	Specifies the filename of the message database file. PLDshell Plus messages are all contained in this file.
MPFFILE	Specifies the name of the Master Parts File
PORT1	Specifies the default serial port used by the APT device programming software.
PORT2-4	Specifies the optional, additional serial ports used by the APT device programming software.
PLD_MLIB	Specifies the macro libraries used during SMF/ADF translation. The default entries are ''TTL.LIB'' and ''EPLDMAC.LIB''.
PARJKTERM	Specifies the value which controls JK product term optimization in the ADF parser. The larger the number, the larger the equation the parser will accept for processing (and the longer the processing will take). Valid values are decimal units (in p-terms)
CMDSHELL	Specifies the command shell program to run.
DEVPROG	Specifies the programming software to use.
EDITOR	Text editor for PLDshell Plus. Default is the DOS 5.0 EDIT editor. Use the **Utilities–Modify Options** menu.

HOTKEYS	Allows selection of menus and submenus by using the first letter of the menu name. The first letter is highlighted. Default is ON. Use the **Utilities–Modify Options** menu.
PRINTER	Sets the printer port. The default is PRN. Use the **Utilities–Modify Options** menu.
PDSEXT	Sets the file extension for PDS files. The default is "PDS". Use the **Utilities–Modify Options** menu.
COMP_ERRFILE YES COMP_RPTFILE YES COMP_EXPANDEQN YES COMP_MINIMIZE YES COMP_AUTOINV YES COMP_FITMODE REASSIGN	These are the Compiler Options, accessible through the Compiler Options submenu. The default option is shown with each option. Alternate entries for COMP_FITMODE are ''NO_ASSIGN'' and ''IGNORE_ASSIGN''.
SIM_ASYNC NO SIM_THRESHOLD 32	These are the Simulator Options, accessible through the Simulation Options submenu. The default is shown with each option.
WAVE_PGLEN 66 WAVE_VIEW GRAHPICAL WAVE_PRINT PLAIN	These are the View Options, accessible through the View menu. The default is shown with each option. The alternate entry for WAVE_VIEW is ''STATETABLE''. The alternate entry for WAVE_PRINT is ''EXTENDED''.
PROGA-PROGX	These are the Run menu program fields, accessible through the Run menu.

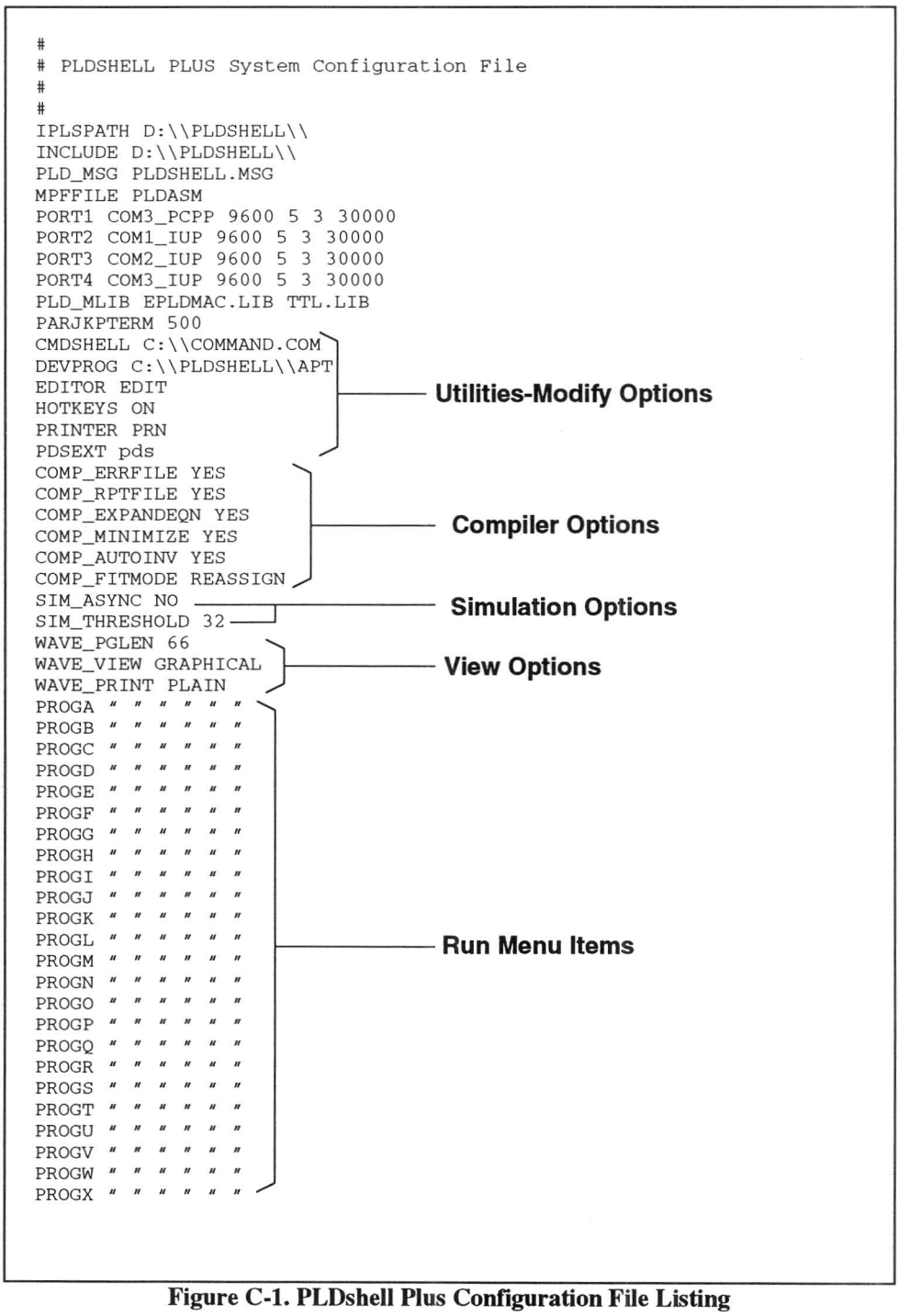
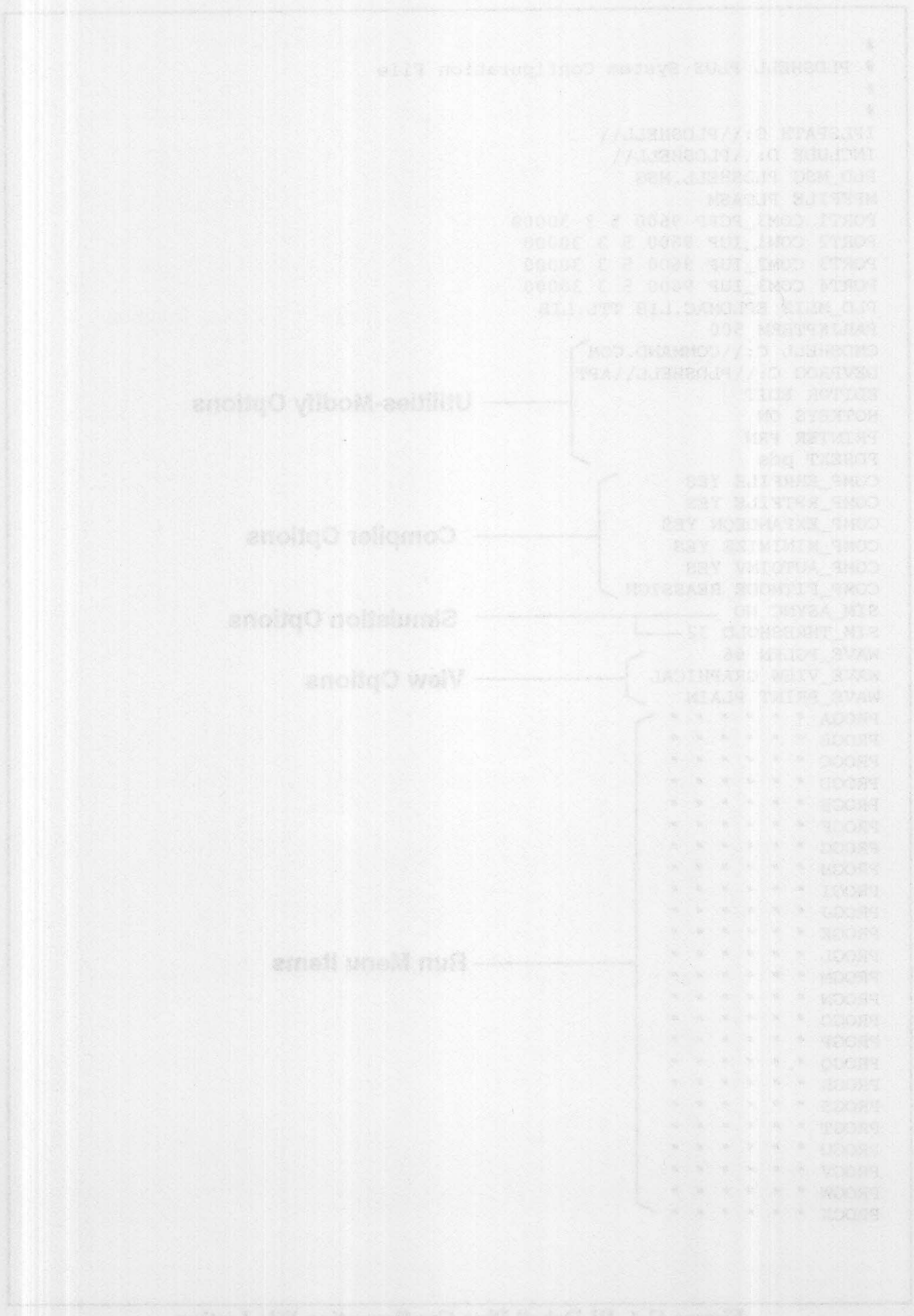


Figure C-1. PLDshell Plus Configuration File Listing



Appendix D – Command Line Interface

The following PLDshell/PLDasm program functions can be directly accessed from DOS:

Compiler
Disassembler
Conversion
Translation

This appendix describes the PLDasm command line interface for these functions.

Command Line Interface

The PLDasm command line syntax is

```
pldasm action [options] files
```

where action is one of the following:

COM[PILE]
D[ISASSEMBLE]
CON[VERT]
T[RANSLATE]

The minimum number of characters for each action is that which identifies it as a unique action (i.e., "D" for Disassemble, "CON" for Convert, etc.).

Compile Commands and Options

To compile a PDS file from the DOS command line use the command syntax:

```
pldasm com [options] filename
```

where the compiler options are:

Table D-1. Command Line Compiler Options

Option	Default	Description
+/– comp	+	Compiler On/Off
+/– sim	+	Simulation On/Off
+/– min	+	Espresso-IImv equation minimizer On/Off
+/– demor	–	Output polarity control On/Off — automatic DeMorganization
+/– xpand	+	Two-level logic expansion On/Off
+/– error	+	Error log file creation On/Off
+/– report	+	Fitter report file creation On/Off
+/– async	–	Show asynchronous events during simulation On/Off
+ thres n	32	Number of asynchronous events, where n is a decimal number in the range of 0 to 32,767
–fit 0	X	Fail if any assigned pins will not fit
–fit 1		Use pin assignments, but reassign if necessary
–fit 2		Ignore all pin assignments in file
-infile	(optional)	Input filename
Where "+" = On and "–" = Off		

Compilation Examples

The following are examples of compiler operation from the DOS command line:

```
pldasm comp -sim -min 4count.pds
```

where simulation and the minimizer are both turned off. No simulation file will be created and p-term minimization will not be performed.

```
pldasm comp -comp +async +thres 55 count.pds
```

where compilation is turned off, and a maximum of 55 asynchronous events will be permitted/shown during each simulation step.

To disassemble a JEDEC file from the DOS command line, use the following command syntax:

```
pldasm dis <part> <package> file
```

where:

<part> is any valid part recognized by PLDasm.

<package> is any valid package type: DIP, PLCC, PGA.

file is a JEDEC input file. The output file is the target Intel device name with a .JED extension.

Disassembly Example

```
pldasm DIS 16V8 PLCC UDCTR.JED
```

where the <part> is a 16V8 in a PLCC package. The input file is UDCTR.JED. Note that the 16V8 JEDEC file will be disassembled into a PDS file for the respective Intel µPLD. The output filename is 85C220.PDS. Table 7-1 lists all supported PLDs.

Convert disassembles a JEDEC file for a PAL device, producing a PDS file, which is then compiled to a JEDEC file for an Intel µPLD. To convert a JEDEC file, use the following command syntax:

```
pldasm CONV <part> <package> [options] file
```

where:

<part> is any valid part recognized by PLDasm.

<package> is any valid package type: DIP, PLCC, PGA.

file is a JEDEC input file. The output file is the target Intel device with a .JED extension.

Conversion Examples

```
pldasm CONV 20L8 DIP UDCNTR.JED
```

where <part> is a 20L8 in a DIP package. The output file is 85C224.JED for an Intel part.

```
pldasm CONV 20L8 DIP +min UDCTR.JED
```

where <part> is a 20L8 in a DIP package. The compiler minimizer is turned on. The output file is 85C224.JED for an Intel part. Table 7-1 lists all supported PLDs.

Translate produces a PDS source file from input files in another PLD language (ADF/SMF files). To use the translate command from the DOS command line, use the following syntax:

```
pldasm TRAN [-i file.in] [-o file.out]
```

where:

-i file.in is the input filename (optional)

-o file.out is the output filename (optional)

Translation Examples

```
pldasm TRAN COUNTER.ADF
```

where the input file COUNTER.ADF is translated into output file CONVERT.PDS

```
pldasm TRAN -i COUNTER.ADF -o NEWCTR.PDS
```

where the input file COUNTER.ADF is translated into output file NEWCTR.PDS.

Appendix E – APT Description

Overview of APT

APT (Advanced Programming Tool) is a software package that allows you to program, read, and verify Intel µPLDs (Erasable Programmable Logic Devices) from a JEDEC file. APT programs devices using the iUP-PC (Universal Programmer Personal Computer) or the iUP-200A/201A Universal Programmers. Both programmers rely on the iUP-GUPI Module Base and programming adaptors. Programming adaptors typically support a family of devices or a specific device in multiple packages. Refer to the *Microcomputer Programmable Logic Handbook* for information on Intel µPLDs. Table E-1 shows Intel µPLD programming support.

Table E-1. PLD Programming Support

Device	iUP-GUPI Adapter	Package Type Supported
iPLD22V10 85C22V10	GUPI LOGIC-V10	24-Pin DIP & 28-PLCC
iPLD610 85C060	GUPI LOGIC-IID	24-Pin DIP *
iPLD910 85C090	GUPI LOGIC-IID	40-Pin DIP *
5C031, EP310	GUPI 20D20J	20-Pin DIP
5C032, EP320	GUPI 20D20J	20-Pin DIP
85C220	GUPI 20D20J	20-Pin DIP & PLCC
85C224	GUPI 28D28J	24-Pin DIP & 28-Pin PLCC
5C060, EP600	GUPI LOGIC-IID	24-Pin DIP
5C090, EP900	GUPI LOGIC-IID	40-Pin DIP
5C180, EP1800	GUPI LOGIC-18	68-Pin PLCC
5C180PGA	GUPI LOGIC-18PGA	68-Pin PGA

APT is operated via a command-line interface. Once invoked, you can perform one of the following operations with a simple command:

- Check for blank device
- Program device from JEDEC file

- Read device to JEDEC file
- Verify device against JEDEC file
- Compute checksum on file
- Change defaults
- Quit

Sample Session: Programming a 85C224 µPLD

This section of the appendix gets you up and running with APT in a short time. Refer to the following sections of this appendix for detailed information on using APT.

The sample session provided here uses a file called 4COUNT.JED. This file is easily created by compiling 4COUNT.PDS using PLDasm. 4COUNT.PDS is shipped with PLDasm. The following programming hardware is required to work through the sample session:

- PCPP or iUP-200A/201A Universal Programmer
- iUP-GUPI Module base
- GUPI 24D28J Programming Adaptor
- Blank 85C224 µPLD

If the programming hardware and device are not available, you can still learn a lot about APT by carefully reading through the sample session.

Make sure that APT and the programming hardware are installed, then proceed as follows:

STEP 1: Invoke APT

APT is invoked from the PLDshell Plus **Program** Menu as shown below. Position the cursor on **Program** and press <Enter> or press <P> for Program. You can also invoke APT from the install directory by typing

APT <Enter>

In either case, the following will be displayed on the screen:

```
APT Release [ x.y ]  SID [ info. ]
Copyright Intel Corporation, <dates>
Welcome to APT

File 'apt.cfg' Does Not Exist, Create?  Y/N  [Yly]: <Enter>

APT>> h
        The help command provides a quick reference to APT commands.
Usage:
?lh[elp] [command] where command is one of the following:
        b[lankcheck]
        c[hecksum]
        d[efaults]
        r[ead]
        p[rogram]
        v[erify]
        q[uit]
        ' \ ' to Escape
```

The prompt for APT is as follows:

```
APT>>
```

STEP 2: Display Session Defaults

Enter the DEFAULTS command to display all session defaults:

```
DEFAULTS <Enter>
Current Defaults for File 'apt.cfg' are:

DESIGNER:
COMPANY:
DATE:
NUMBER:
REVISION:
EPLD:
COMMENT:

        JEDEC file currently in memory      = <none>.jed
        Device name                         = <none>
        Intel Part name                     = <none>
        Communication Port                  = PORT1
        XOR Map File                        = <none>.xor
        Save Log Messages                   = YES
        Repeat Count                        = 1000
```

STEP 3: Change Defaults

Enter the DEFAULTS command to change the port number to "Port 3" :

```
DEFAULTS -c PORT3 <Enter>
Current Defaults for File 'apt.cfg' are:

DESIGNER:
COMPANY:
DATE:
NUMBER:
REVISION:
EPLD:
COMMENT:

        JEDEC file currently in memory      = <none>.jed
        Device name                         = <none>
        Intel Part name                     = <none>
        Communication Port                  = PORT3
        XOR Map File                        = <none>.xor
        Save Log Messages                   = YES
        Repeat Count                        = 1000
```

STEP 4: Insert the Device and Execute BLANKCHECK

Insert the 85C224 into the 24-pin DIP socket on the 24D28J Adaptor. Secure the device with the lever. Enter the BLANKCHECK command as shown below. Note that since this is the first time that the programmer is accessed since invocation, APT initializes the programmer before executing the blankcheck. Also, since no device-specific software (.DSS) files have previously been loaded, the 85C224.DSS file is also loaded.

```
BLANKCHECK -D 85C224 <Enter>

Initializing Serial Port COM3_PCPP
Initializing Device 85c224
Initializing PCPP
PCPP Version:  PCPP EPROM Loader V1.0, 08-05-86, 14:25
Copyright Intel Corporation, 1986
Running Diagnostics on PCPP
Downloading 'c:\iplsii\pcpp85.obj'
Downloading 'c:\iplsii\85c224.dss'
Building Physical Data Base for 85c224

Insert the 85c224 device and press return to continue or '\' to Escape: <Enter>
        Device is Blank

Insert the 85c224 device and press return to continue or '\' to Escape: \ <Enter>
```

Use a backslash (\) followed by <Enter> to exit the blankcheck loop (repeat count is set to loop for 1,000 devices).

STEP 5: Program Device

Program the device using 4COUNT.JED and using the defaults for all other options:

```
PROGRAM 4COUNT.JED <Enter>

Reading JEDEC Fuse Data from '4COUNT.jed' at address XXXX
Computed fusesum = YYYYH, file fusesum = ZZZZH
Computed filesum = YYYYH, file filesum = ZZZZH

Finished Reading JEDEC file '4COUNT.jed'.
        Turbo is ON
        Verify Protect is OFF

Insert the 85c224 device and press return to continue or '\' to Escape:  <Enter>

Programming Device Address XXXXH – YYYYH
Programming Completed
Verifying Device
Verifying Device Address XXXXH – YYYYH
Finished Verifying Device, Errors = 0

Insert the 85c224 device and press return to continue or '\' to Escape:  \ <Enter>
```

Use a backslash (\) followed by <Enter> to exit the program loop (repeat count is set to loop for 1,000 devices).

STEP 6: Read Device Contents into JEDEC File

Execute the READ command to read the contents of the device, storing the contents in a file named NEWFILE.JED:

READ <Enter>
Enter a JEDEC File Name or '\' to Escape:
NEWFILE.JED <Enter>

Insert the 85c224 device and press return to continue or '\' to Escape: **<Enter>**

Reading Device Address XXXXH – YYYYH
Finished Reading Device
Turbo is ON
Writing JEDEC Fuse Data into 'NEWFILE.jed' at address XXXX
Finished Writing JEDEC file 'NEWFILE.jed'.

STEP 7: Verify Device Against JEDEC File

Execute the VERIFY command to check the contents of the device against the JEDEC file NEWFILE:

V NEWFILE.JED <Enter>

Reading JEDEC Fuse Data from 'NEWFILE.jed' at address XXXX
Computed fusesum = YYYYH, file fusesum = ZZZZH
Computed filesum = YYYYH, file filesum = ZZZZH
Finished Reading JEDEC file 'NEWFILE.jed'.
Turbo is ON

Insert the 85C224 device and press return to continue or '\' to Escape: **<Enter>**
Verifying Device Address XXXXH – YYYYH
Finished Verifying Device, Errors = 0

Insert the 85c224 device and press return to continue or '\' to Escape: **\ <Enter>**

STEP 8: Exit to PLDshell Plus

Execute the QUIT command to exit to the OS/command shell:

```
QUIT <Enter>
EXITING APT
```

APT Files

As shown in Figure E-1, APT accepts a .JED file, a .JPM file, and a .CFG file as input and outputs two optional files: a .LOG file and a .XOR file. One additional file, a .DSS file is required for each device type. The following list describes each of these files:

- .JED (JEDEC) files are typically produced using Intel's PLDasm compiler, which is included in PLDshell Plus. Other programs may also be used to create the JEDEC file.

- .JPM (JEDEC to Physical Map) files for each device contain the information for mapping the JEDEC addresses into the physical row and column locations for the specified device. The proper JPM is required for programming, reading, or verifying devices. These files are provided with APT and are maintained by Intel.

- A .CFG file (Defaults Configuration File) contains session defaults that define the default programming conditions. The .CFG file is read at invocation to automatically set session variables.

- .LOG files are generated by APT under user control. These files contain all user messages and commands, including error messages. Error message help information is available on-line in PLDshell Plus. Use the View Error/Log File menu options, move to the error message, and press <F10> to obtain this help information.

- .XOR files are generated by APT under user control. An .XOR (Exclusive-OR) file contains the mismatched device/JEDEC file data and addresses from a Verify operation (if any mismatches are detected). The user is prompted to save the mismatched data in an .XOR file.

- .DSS (device-specific software) files, shipped with the GUPI adaptors, contain device-specific programming algorithms and tables. These files are required to program Intel µPLDs and are maintained by Intel.

APT can be invoked directly from the Operating System or PLDshell Plus to program, read, and verify Intel µPLDs. This section describes APT invocation, general interface conventions, error message, and filename conventions.

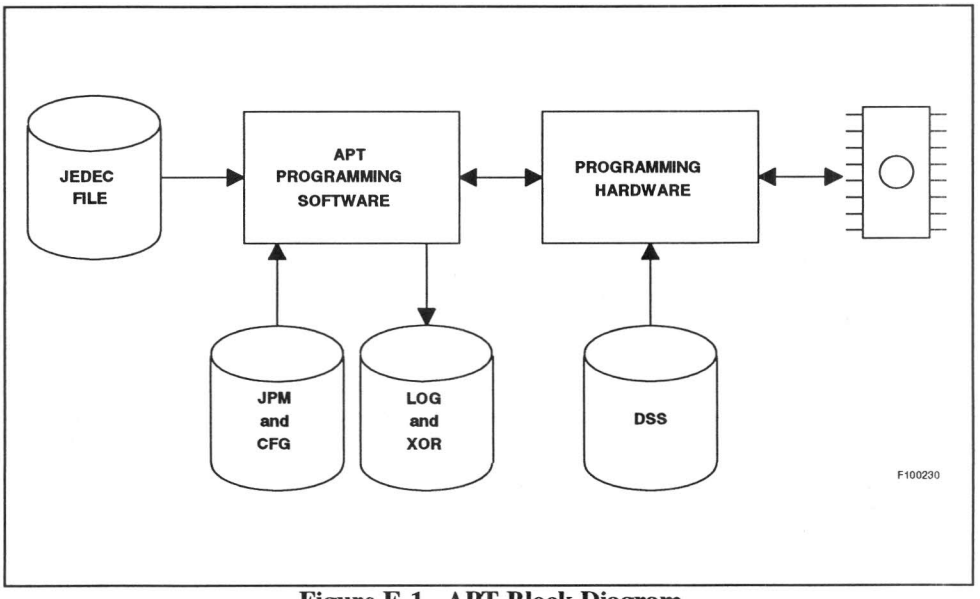


Figure E-1. APT Block Diagram

Command Line Invocation

The prompt for APT is as follows:

```
APT>>
```

APT command lines consist of a required command name, optionally followed by arguments. Arguments can be filenames and/or command options. The following command line shows this structure:

```
PROGRAM  MYJED.JED  -d  85C22V10  -r  3 <Enter>
```

where PROGRAM is the command, MYJED.JED is a JEDEC file name, -d 85C22V10 sets the device type, -r 3 instructs APT to perform the operation three times, and <Enter> terminates the command line. This command would be used to program three 85C22V10 devices using the contents of MYJED.JED. All APT commands end with an <Enter>.

NOTE

The backslash character ' \' followed by <Enter> is used to escape from the current APT prompt. This sequence performs the same function as the ESC key in most software applications.

Session Defaults

When no files or options are specified on the command line, APT uses session default values whenever possible. When no default values are detected, the user is prompted for the required information. Some session defaults are read from a Defaults Configuration File (.CFG). Other session defaults must be set by the user after invocation.

Session defaults are divided into two groups, JEDEC header defaults, and APT execution defaults. The Defaults Configuration File (.CFG file) contains all of the header defaults and some of the execution defaults.

JEDEC header defaults provide header information for JEDEC files created by the read command from pre-programmed devices. These defaults *cannot* be changed by using the DEFAULTS command. They can only be changed by editing the APT.CFG file (a user-defined .CFG file) with an ASCII text editor. Examples of these defaults are as follows:

```
DESIGNER: Your Name
COMPANY: Your Company
DATE: Current Date
NUMBER: Final Eng. Part No.
REVISION: 1.0
EPLD: 85c224
COMMENT: This is a comment to help document the design. It can span more than
one line.
```

Execution defaults, which are used when executing APT commands, are shown in the following example:

```
JEDEC file currently in memory      = <none>.jed
Device name                         = 85c224
Intel Part name                     = 85c224
Communication Port                  = PORT3
XOR Map File                        = <none>.xor
Save Log Messages                   = YES
Repeat Count                        = 1000
```

These defaults can be changed by command line options for several APT commands. Two defaults (Communication Port and Save Log Messages) are stored in the .CFG file and read on invocation. When exiting APT, the user is given the option to write these defaults (if they have been changed) to the .CFG file for later use. These two defaults can also be changed by editing the .CFG file using an ASCII text editor.

APT Error Messages

APT displays error messages when invalid conditions are detected. Warning messages are displayed when potentially harmful situations are detected (e.g., overwriting information, etc.). Error and warning messages are numbered, and are formatted as follows:

```
ERROR E586-APT: Unrecognized APT command, try again.
```

Error message help information is available on-line in PLDshell Plus. Use the View Error/Log File menu options, move to the error message, and press <F10> to obtain this help information. The help information describes the probable causes for each message, and recommended action to correct the error condition.

Filename Conventions

APT filenames follow the conventions appropriate for the respective operating system as far as pathnames, character length, upper/lower case distinction, etc. The same extensions, however, are followed across all platforms/operating systems:

.JED	JEDEC programming file
.LOG	Session log file (all message and user entries)
.XOR	Exclusive-OR file (results of Verify)
.CFG	Defaults Configuration File
.DSS	Device-Specific Software file (programming algorithms and parameters)
.JPM	JEDEC to Physical Map file

APT Commands

This section presents all commands in alphabetical order, with each command section containing a discussion of the command, and showing syntax, abbreviated form, and examples. Table E-1 lists and summarizes all APT commands and arguments.

Command Arguments

The following list shows all APT command options and arguments and describes their use. Each command accepts a different subset of arguments. Some commands do not accept any arguments.

jedfile	JEDEC programming file. Contains the bit-image used to program devices. Ends with a .JED extension.
file	Defaults Configuration File. Contains default names and option values used during programming. Ends with a .CFG extension.
-c port#	Specify serial communication port in the system configuration file. Used to change the serial port for the current session. ''port'' is a keyword and may not be abbreviated. # is a decimal number from 1 to 8.
-d device	Specify device name. The name must be a valid Intel µPLD part name (i.e., iPLD610, 5AC324, iPLD22V10, 85C508, 85C22V10, etc.).
-l | +l	Save/Don't Save log File. A +l (or +L) instructs APT to save all messages and user commands in an ASCII file named apt.LOG. -l (or -L) turns the save option off. When specified on the command line, this option overrides/updates the session default loaded from the .CFG file. The name of the log file is the same as the JEDEC filename with a .LOG extension.
-p | +p	Set/Clear Security Bit (Verify Protect Bit). A +p (or +P) sets the Security Bit during programming to prevent subsequent device reads. A -p (or -P) keeps the Security Bit cleared during programming to allow subsequent device reads. When specified on the command line, this option overrides the value in the JEDEC file.
-r count	Set repeat count value. Sets the number of times subsequent APT commands are looped. Useful for programming several devices in a row. The count field is a decimal number from 1 to 1,000. The default is 1,000.

-t | +t — Set/Clear Turbo Bit. A +t (+T) sets the Turbo Bit during programming (TURBO=ON) to allow the device to run at full speed (no power-down). A -t (-T) keeps the Turbo Bit cleared during programming (TURBO=OFF) to allow the device to enter standby mode between input transitions.

E-2. APT Command Summary

Command	Arguments	Description
BLANKCHECK	-d device -r count	Checks for blank device
CHECKSUM	jedfile	Computes checksums from JEDEC file.
DEFAULTS	file -d device -c port# +l | -l -r count	Displays/changes defaults configuration (.CFG) file and/or session defaults.
EXIT		Exits to operating system/shell
HELP or ?		Displays help information for specified command.
PROGRAM	jedfile -d device -c port# -r count +p | -p +t | -t	Programs device from JEDEC file.
QUIT		Exits to operating system/shell
READ	jedfile -d device -c port#	Reads device contents to JEDEC file.
VERIFY	jedfile -d device -c port# -r count	Compares device contents agains JEDEC file.
\		Escapes from current context back to APT prompt.
Ctrl C		"Ctrl + C" terminates all processing and returns control to operating system/shell.

BLANKCHECK

Check for blank/erased device(s)

Syntax

BLANKCHECK [options]

Abbreviated Form

B . . . BLANKCHECK

Discussion

This command checks to see if the specified device is blank (EPROM cells in erased state). The command returns a ''Device is Blank'' or ''Device is Not Blank'' message depending on the state of the device. If no device is specified in the command line and no session default has been previously specified, the user is prompted for the name of a device. The specified device must be properly inserted in the socket on the programming adaptor before executing this command. The command can be repeated by specifying a count value on the command line. The default count is used if no count value is specified. Valid options are as follows:

-d device specify *device*
-r count *count* times

Note that for some devices, the blank/erased state is all zeros while for other devices it is all ones. APT uses the proper value for the specified device.

Examples

1. Use full form of command:

```
BLANKCHECK -d 5AC324 <Enter>
Insert the 5ac324 device and press return to continue or '\' to Escape: <Enter>
        Device is Blank
Insert the 5ac324 device and press return to continue or '\' to Escape: \ <Enter>
```

2. Use abbreviated form and specify count value on command line:

```
BLANK -d 85C508 -r 2 <Enter>
Insert the 85c508 device and press return to continue or '\' to Escape: <Enter>
        Device is Blank
Insert the 85c508 device and press return to continue or '\' to Escape: <Enter>
        Device is Blank
```

3. Use abbreviated form with no device on command line:

BL <Enter>
Illegal or Unspecified Device.
Enter a Default type or '\' to Escape: **85C224 <Enter>**
Insert the 85c224 device and press return to continue or '\' to Escape: **<Enter>**
Device is Blank
Insert the 85c224 device and press return to continue or '\' to Escape: **\ <Enter>**

CHECKSUM

Generates checksums for JEDEC file

Syntax

CHECKSUM [jedfile]

Abbreviated Form

C . . . CHECKSUM

Discussion

This command reads the specified JEDEC file and generates transmission checksums on the bytes in the file and in the fuse section. Both the computed checksum and the checksum in the JEDEC file are displayed for the fuse section and the file. If either checksum has changed, you are prompted whether or not to update the JEDEC file with the computed checksum. If the JEDEC filename is not specified on the command line and no JEDEC file is currently in memory, the user is prompted for the filename. This command does not load the JEDEC file into program memory; therefore, the JEDEC filename in the session defaults is not updated.

Examples

1. Specify filename on command line:

```
CHECKSUM  MYPROGRM.JED <Enter>
Finished Reading JEDEC file 'myprogrm'.
Computed fusesum = YYYYH, file fusesum = ZZZZH
Computed filesum = YYYYH, file filesum = ZZZZH
```

2. Use abbreviated command and specify filename on command line:

```
C NEWPROGR.JED <Enter>
Finished Reading JEDEC file 'newprogr'.
Computed fusesum = YYYYH, file fusesum = ZZZZH
Computed filesum = YYYYH, file filesum = ZZZZH
```

3. Use abbreviated form and prompt for filename (checksum does not match file):

CHECK <Enter>
Please Enter a JEDEC File Name (no .jed extension):
NEWPROGR <Enter>

Finished Reading JEDEC file 'newprogr'.
Computed fusesum = YYYYH, file fusesum = ZZZZH
Computed filesum = YYYYH, file filesum = ZZZZH
Computed Checksum does not match filesum, update? Y/N [Yly] **Y <Enter>**
Checksum(s) for file 'newprogr' have been updated.
Computed checksums(s) not equal to filesum(s), update file? Y/N [Nln] **Y<Enter>**

DEFAULTS

Display/change session defaults/defaults configuration file

Syntax

DEFAULTS [file] [options]

Abbreviated Form

D . . . DEFAULTS

Discussion

When entered with a filename only, this command reads the specified Defaults Configuration File and displays the current set of defaults. If the specified file is not found, APT prompts you to create it and to supply default values.

If no filename is specified on the command line, the .CFG file previously used is loaded. If no previous file was loaded, the default file (APT.CFG) is used.

If APT.CFG is not present, you are prompted to create it and to supply default values.

When entered with options, the session defaults are updated. When you exit APT after changing the communications port or log file status, you are prompted to write these values to the Defaults Configuration File. Valid options are as follows:

-d device	override default *device* (not written to Defaults Configuration File)
-c port#	serial communication port # (1 to 8)
+l | -l	save (+) or don't save (-) Log File
-r count	repeat command *count* times (not written to Defaults Configuration File)

NOTES

Default Configuration Files contain more information than can be changed via the DEFAULTS command. Any stored value can be changed by editing the file with a standard ASCII text editor. Refer to "Defaults Configuration File" for details.

Session defaults for device name can be changed via the DEFAULTS command, but this name is not stored in a Defaults Configuration File. Device names are automatically updated whenever a valid PART: field is read from a JEDEC file.

Examples

1. Display current defaults:

   D <Enter>
   Current Defaults for File 'apt.cfg' are:

   DESIGNER: Your Name
   COMPANY: Your Company
   DATE: Current Date
   NUMBER: Final Eng. Part No.
   REVISION: 1.0
   EPLD: 85C224
   COMMENT: This is a comment to help document the design. It can span more than one line.

JEDEC file currently in memory	= <none>.jed
Device name	= <none>
Intel Part name	= <none>
Communication Port	= PORT3
XOR Map File	= <none>.xor
Save Log Messages	= YES
Repeat Count	= 1000

2. Change file to disable logging of session messages/commands:

   D -L <Enter>
   Current Defaults for File 'apt.cfg' are:

   DESIGNER: Your Name
   COMPANY: Your Company
   DATE: Current Date
   NUMBER: Final Eng. Part No.
   REVISION: 1.0
   EPLD: 85C224
   COMMENT: This is a comment to help document the design. It can span more than one line.

JEDEC file currently in memory	= <none>.jed
Device name	= <none>
Intel Part name	= <none>
Communication Port	= PORT3
XOR Map File	= <none>.xor
Save Log Messages	= NO
Repeat Count	= 1000

EXIT

Exit APT and return to operating system or shell

Syntax

EXIT

Abbreviated Form

E . . . EXIT or X

Discussion

This command exits APT and returns to the operating system or shell. Temporary files are deleted and open files are closed. (See also: QUIT).

Examples

1. Use full form:

   EXIT <Enter>
   EXITING APT

2. Use abbreviated form:

   E <Enter>
   EXITING APT

3. Exit after stored defaults have been changed, saving the new defaults:

   X <Enter>
   EXITING APT

   Defaults Have Been Modified, Save as 'apt.cfg'? Y/N [Yly]: **y <Enter>**
   File 'apt.cfg' has been updated.

HELP

Display help information for commands

Syntax

HELP [command]

Abbreviated Form

H . . . HELP or ?

Discussion

Displays a brief description of the specified command, including usage.

Examples

1. Use the full form to get help on program command:

HELP P <Enter>

The program command programs a device from data in a JEDEC file.

Usage:

p [program] [jedfile] [-d device] [-c port#] [+t | -t] [+p | -p] [-r count]

2. Use abbreviated form to list all commands:

H <Enter>

The help command provides a quick reference to APT commands.

Usage:

?| h[elp] [command] where command is one of the following:
b[lankcheck]
c[hecksum]
d[efaults]
r[ead]
p[rogram]
v[erify]
q[uit]
' \' to Escape

PROGRAM

Program device from JEDEC file

Syntax

PROGRAM [jedfile] [options]

Abbreviated Form

P . . . PROGRAM

Discussion

This command reads the specified JEDEC file and programs the specified device. If a JEDEC file and/or device are not specified on the command line, APT uses the session defaults. If no session defaults are currently in memory, APT prompts you for them.

Valid options are as follows:

-d device	specify *device*
-c port#	serial communication port # (1-8)
-r count	repeat command *count* times
+p | -p	set (+) or don't set (-) Verify Protect Bit
+t | -t	set (+) or don't set (-) Turbo Bit

Examples

1. Program a 85C224 immediately after invocation, specifying JEDEC file and using defaults for other options:

```
PROGRAM 4COUNT.JED +t -p <Enter>
Initializing Serial Port COM3_PCPP
Initializing Device 85c224
Initializing PCPP
PCPP Version:  PCPP EPROM Loader <version information>
Copyright Intel Corporation, 1986
Running Diagnostics on PCPP
Downloading 'c:\iplsii\pcpp85.obj'
Downloading 'c:\iplsii\85c224.dss'
Building Physical Data Base for 85c224
Reading JEDEC Fuse Data from '4COUNT.jed' at address XXXX
Computed fusesum = YYYYH, file fusesum = ZZZZH
Computed filesum = YYYYH, file filesum = ZZZZH
Finished Reading JEDEC file '4COUNT.jed'.
```

Turbo is ON
Verify Protect is OFF

Insert the 85c224 device and press return to continue or '\' to Escape: **<Enter>**

Programming Device Address XXXXH – YYYYH
Programming Completed
Verifying Device
Verifying Device Address XXXXH – YYYYH
Finished Verifying Device, Errors = 0

Insert the 85c224 device and press return to continue or '\' to Escape: **\ <Enter>**

2. Program a 5AC324 as part of a sequence of several identical devices (JEDEC file is already loaded and one 5AC324 has already been programmed with +t and +p).

P <Enter>
Turbo is ON
Verify Protect is ON

Insert the 5ac324 device and press return to continue or '\' to Escape: **<Enter>**

Programming Device Address XXXXH – YYYYH
Programming Completed
Verifying Device
Verifying Device Address XXXXH – YYYYH
Finished Verifying Device, Errors = 0
Locking Device
Device is Locked

Insert the 5ac324 device and press return to continue or '\' to Escape: **\ <Enter>**

3. Program three 85C508s after a 85C224 device has been programmed:

```
PROGRAM -D 85C508 SAMP3 -R 3 +P -T <Enter>
Downloading 'c:\iplsii\85c508.dss'
Building Physical Data Base for 85c508
Reading JEDEC Fuse Data from 'SAMP3.jed' at address XXXX
Computed fusesum = YYYYH, file fusesum = ZZZZH
Computed filesum = YYYYH, file filesum = ZZZZH
Finished Reading JEDEC file 'SAMP3.jed'.
        Turbo is OFF
        Verify Protect is ON

Insert the 85c508 device and press return to continue or '\' to Escape: <Enter>

Programming Device Address XXXXH – YYYYH
Programming Completed
Verifying Device
Verifying Device Address XXXXH – YYYYH
Finished Verifying Device, Errors = 0
Locking Device
        Device is Locked

Insert the 85c508 device and press return to continue or '\' to Escape: <Enter>

Programming Device Address XXXXH – YYYYH
Programming Completed
Verifying Device
Verifying Device Address XXXXH – YYYYH
Finished Verifying Device, Errors = 0
Locking Device
        Device is Locked

Insert the 85c508 device and press return to continue or '\' to Escape: <Enter>

Programming Device Address XXXXH – YYYYH
Programming Completed
Verifying Device
Verifying Device Address XXXXH – YYYYH
Finished Verifying Device, Errors = 0
Locking Device
        Device is Locked

Insert the 85c508 device and press return to continue or '\' to Escape: \ <Enter>
```

QUIT

Exit APT and return to operating system or shell

Syntax

QUIT

Abbreviated Form

Q . . . QUIT or X

Discussion

This command exits APT and returns to the operating system or shell. Temporary files are deleted and open files are closed.

Examples

1. Use full form:

   QUIT <Enter>
   EXITING APT

2. Use abbreviated form:

   Q <Enter>
   EXITING APT

3. Quit after stored defaults have been changed, saving the new defaults:

   X <Enter>
   EXITING APT
   Defaults Have Been Modified, Save as 'apt.cfg'? Y/N [Yly]: **y <Enter>**
   File 'apt.cfg' has been updated.

READ

Reads contents of device and stores in JEDEC file

Syntax

READ [jedfile] [options]

Abbreviated Form

R . . . READ

Discussion

This command reads the specified device and stores the contents in a JEDEC file. If a device is not specified on the command line, APT uses the previously specified session default. If a default has not been entered, the user is prompted for a device name. If a JEDEC file is not specified, the user is prompted for a filename.

Valid options are as follows:

-d device specify *device*

-c port# serial communication port# (1-8)

Examples

1. Read device and store in JEDEC file.

```
READ -d 85C224 <Enter>
Enter a JEDEC File Name or '\' to Escape:
READSAMP.JED <Enter>

Insert the 85c224 device and press return to continue or '\' to Escape: <Enter>

Reading Device Address XXXXH – YYYYH
Finished Reading Device
        Turbo is ON
Writing JEDEC Fuse Data into 'READSAMP.jed' at address XXXX
Finished Writing JEDEC file 'READSAMP.jed'.
```

2. Read device, specifying device and filename on command line:

   R NEWSAMP -d 5AC324 <Enter>

   Insert the 5ac324 device and press return to continue or '\' to Escape: **<Enter>**

   Reading Device Address XXXXH – YYYYH
   Finished Reading Device
   Turbo is ON
   Writing JEDEC Fuse Data into 'READSAMP.jed' at address XXXX
   Finished Writing JEDEC file 'NEWSAMP.jed'.

VERIFY

Read contents of device and compare against JEDEC file

Syntax

VERIFY [jedfile] [options]

Abbreviated Form

V . . . VERIFY

Discussion

This command reads the specified device and compares the device contents with the specified JEDEC file. If a device and/or JEDEC file are not specified on the command line, APT uses the session defaults. If defaults are not found there, the user is prompted for this information. If the data read from the device does not match the programming image in the JEDEC file, a message is displayed and the user is prompted to store the differences in a file. This file, which is identified by a .XOR extension, is the Exclusive-OR of the comparison.

Valid options are as follows:

-d device	specify *device*
-c port#	serial communication port # (valid numbers are 1-3)
-r count	repeat command *count* times

Examples

1. Verify a device using the default values:

```
VERIFY <Enter>
Enter a JEDEC File Name or '\' to Escape:
NEWFILE.JED

Reading JEDEC Fuse Data from 'NEWFILE.jed' at address XXXX
Computed fusesum = YYYYH, file fusesum = ZZZZH
Computed filesum = YYYYH, file filesum = ZZZZH
Finished Reading JEDEC file 'NEWFILE.jed'.
        Turbo is ON
Insert the 85c224 device and press return to continue or '\' to Escape:  <Enter>
Verifying Device Address XXXXH – YYYYH
Finished Verifying Device, Errors = 0

Insert the 85c224 device and press return to continue or '\' to Escape:  \ <Enter>
```

2. Verify the specified device against the specified JEDEC file:

```
V -d 85C508 NEW.JED <Enter>
Reading JEDEC Fuse Data from 'NEWFILE.jed' at address XXXX
Computed fusesum = YYYYH, file fusesum = ZZZZH
Computed filesum = YYYYH, file filesum = ZZZZH
Finished Reading JEDEC file 'NEW.jed'.

Insert the 85c508 device and press return to continue or '\' to Escape:  <Enter>

Verifying Device Address XXXXH – YYYYH
Finished Verifying Device, Errors = 0

Insert the 85c508 device and press return to continue or '\' to Escape:  \ <Enter>
```

3. Verify the specified device against the specified JEDEC file immediately after invocation. In this case, errors are detected during the verify. The user answers Y <Enter> when prompted to save the Verify errors in ERRORS.XOR:

```
V -d 85C224 CONTROL <Enter>
Initializing Serial Port COM3_PCPP
Initializing Device 85c224
Initializing PCPP
PCPP Version:  PCPP EPROM Loader <version information>
Copyright Intel Corporation, 1986
Running Diagnostics on PCPP
Downloading 'c:\iplsii\pcpp85.obj'
Downloading 'c:\iplsii\85c224.dss'
Building Physical Data Base for 85c224
Reading JEDEC Fuse Data from 'NEWFILE.jed' at address XXXX
Computed fusesum = YYYYH, file fusesum = ZZZZH
Computed filesum = YYYYH, file filesum = ZZZZH
Finished Reading JEDEC file 'CONTROL.jed'.

Insert the 85c224 device and press return to continue or '\' to Escape:  <Enter>

Verifying Device Address XXXXH – YYYYH
Errors Exist, do you want to continue?  Y/N  [Yly] :  Y <Enter>
Saving Verify Errors in file, errors.xor
Addr 0600H                                File 02H Device 00HXOR 02H
Addr 0602H                                File 02H Device 00HXOR 02H
Addr 0604H                                File 02H Device 00HXOR 02H
Addr 0606H                                File 02H Device 00HXOR 02H
Addr 0608H                                File 02H Device 00HXOR 02H
Addr 060aH                                File 02H Device 00HXOR 02H
```

(Verify error messages are displayed on the same line; the current message overwrites the previous message.)

```
Finished Verifying Device, Errors = 6
        Turbo is ON

Insert the 85c224 device and press return to continue or '\' to Escape:  \ <Enter>
```

Defaults/Configuration File

This chapter describes the format of the Defaults Configuration (.CFG) file used by APT so that users may alter parameters as desired. Figure E-2 shows the format for APT.CFG as installed in your working directory. Note that all fields except SAVELOG and SIOPORT are blank. Changes can be made to any field by an ASCII text editor. Figure E-3 shows a modified APT.CFG FILE.

The first eight fields are comments and must be enclosed in quotation marks. The maximum number of characters for all fields (except the comment field) is 80. A maximum of 512 characters may appear in the comment field. The comment fields interprets a backslash '\' followed by a carriage return simply as a carriage return. Initial spaces in a comment field are not displayed, except when they appear after the '\' + carriage return sequence.

A YES or NO (uppercase or lowercase) are the only valid options for SAVELOG field.

SIOPORT options are PORT1 through PORT8 (uppercase or lowercase). Note that the port name in the APT.CFG file is a logical port name only and may not reflect a numerical port. The PLDSHELL.CFG file maps logical APT ports to physical communications ports, as follows:

```
PORT1  COM3_PCPP  9600  5  250  2000
PORT2  COM1_IUP   9600  5  2    5000
PORT3  COM2_IUP   9600  5  2    5000
PORT4  COM2_IUP   9600  5  2    5000
```

PORT1 is the default port for APT. (The port description specifies the physical DOS port and key interface parameters.

```
DESIGNER: " "
COMPANY: " "
DATE: " "
NUMBER: " "
REVISION: " "
EPLD: " "
COMMENT: " "
XORFILE: " "
SAVELOG:  YES
SIOPORT:  PORT3
```

Figure E-2. APT.CFG File Before Alternation

```
DESIGNER:  "Joe Designer"
COMPANY:  "New Wave Electronics"
DATE:  "6/29/91"
NUMBER:  "123456-789"
REVISION:  "B"
EPLD:  "85C508"
COMMENT:  "Main Memory Decoder for i860 System"
XORFILE:  "TEMP.XOR"
SAVELOG:  YES
SIOPORT:  PORT3
```

Figure E-3. APT.CFG File After Alteration

Appendix F – Basic PLD Information

This section provides basic information for designers who may be new to PLDs (Programmable Logic Devices). The topics include:

- What are PLDs
- Basic architecture of PLDs
- Why use PLDs
- PLD design process

What Are PLDs?

PLDs are digital devices that can be configured by the user to implement a wide variety of logic functions in systems. As shown in Figure F-1, PLDs have input pins, a programmable logic array, and I/O pins. Many PLDs have programmable outputs that increase their flexibility and thus make them suited for a wider variety of applications than PLDs with fixed outputs.

Basic Architecture of PLDs

PLD inputs feed the logic array, which are made up of columns and rows. Figure F-2 shows such an array. Each column pair reflects the true and complement states of an input. Each row constitutes an AND term (also called a "product term", or "p-term" for short). Logical connections are established between different columns and rows in the array to determine which combination of inputs will drive the p-term (AND term) high. In Intel's CMOS PLDs, this connection is established by EPROM cells. Other methods of making connections are used in PLDs based on different technology.

More than one p-term typically feeds an OR gate, which in turn drives the fixed or configurable output. This summing of product terms is often referred to by the abbreviation SOP (Sum of Products). Figure F-3 shows an SOP input, followed by an invert select bit and a fixed combinatorial output buffer. To minimize the need to feed output signals back to input pins, internal paths back to the logic array are usually provided (feedbacks).

Other output options may include the ability to select a registered versus a combinatorial output, and an output enable control (one or more p-terms). Registered outputs may contain preset and clear signals (one or more p-terms) or asynchronous clock signals (one or more p-terms). Figure F-4 shows a registered output with a p-term to control its output enable (OE).

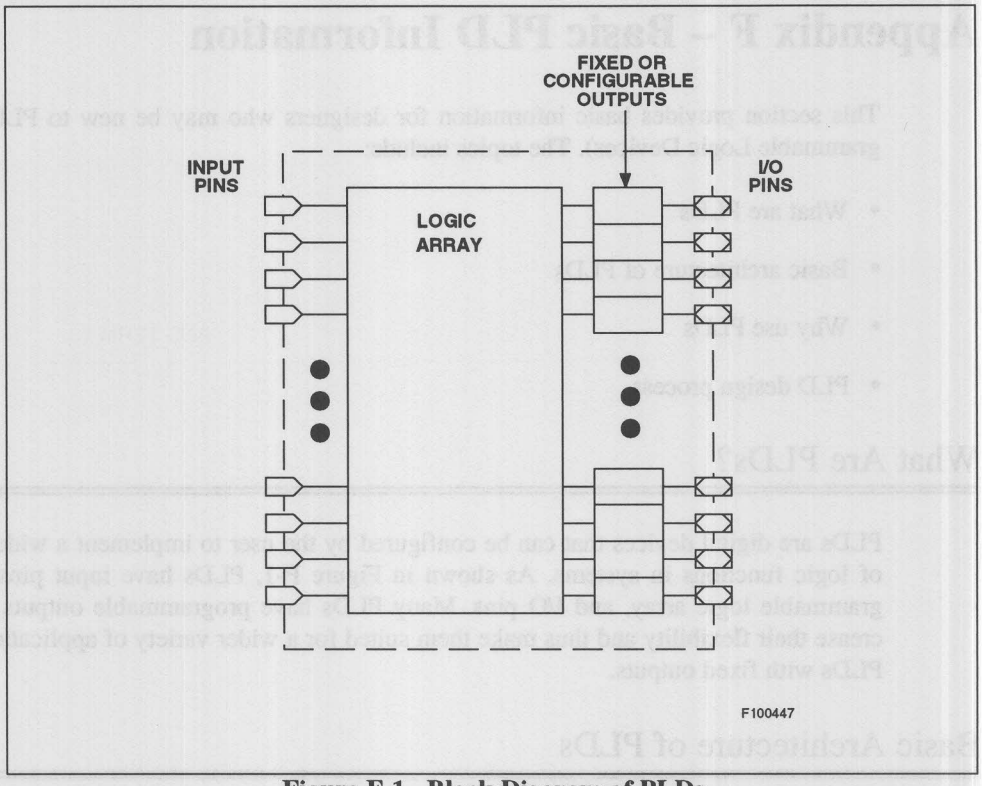


Figure F-1. Block Diagram of PLDs

Why Use PLDs?

The flexibility and programmability of PLDs make designing with them much quicker than designing with discrete logic. The ability to customize PLDs for a specific application allows a few general purpose PLDs to implement most functions once reserved for the hundreds of devices in the ''7400'' series logic family. Use of PLDs results in a component stock reduction in engineering and manufacturing. PLDs also take up less space on printed circuit boards than discrete devices because more logic functions can be fit into each PLD than with discrete devices.

Once the decision to move from discrete logic to PLDs is made, the next question is: ''Which PLDs?'' The more flexible a PLD is, the more useful it is because a designer can fit more logic into it. Since Intel µPLDs are *supersets* of common industry-standard PLD architectures (additional flexibility), use of Intel µPLDs provides designers with an ''added edge'' over other devices on the market. Chapter 7 of this guide discusses these superset features in greater detail.

Intel µPLDs are manufactured using Intel's patented CHMOS III semiconductor process with EPROM cell technology that has proven itself over 20 million memory devices

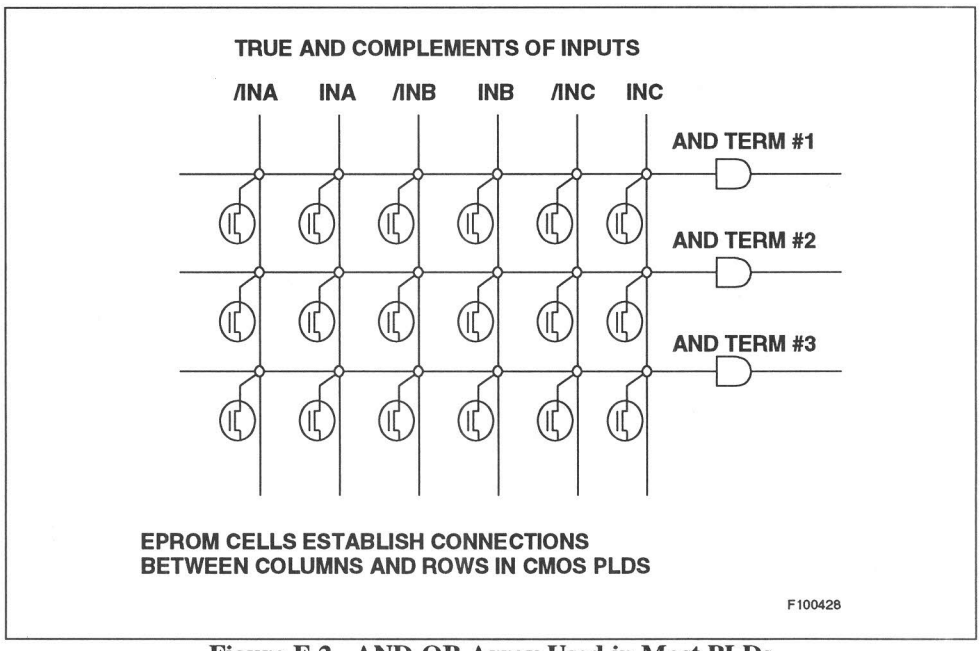


Figure F-2. AND-OR Array Used in Most PLDs

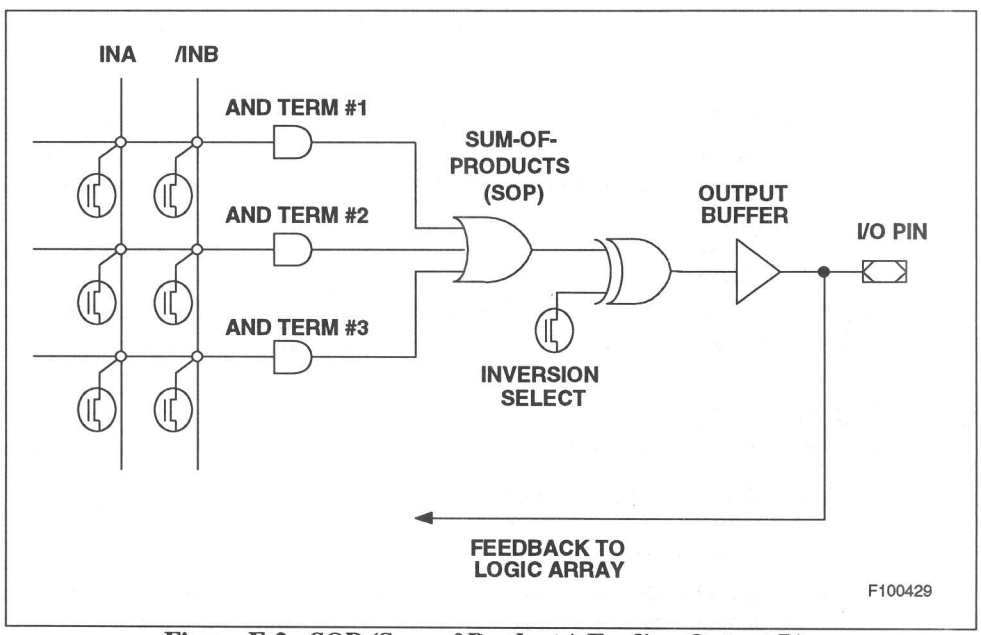


Figure F-3. SOP (Sum of Products) Feeding Output Pin

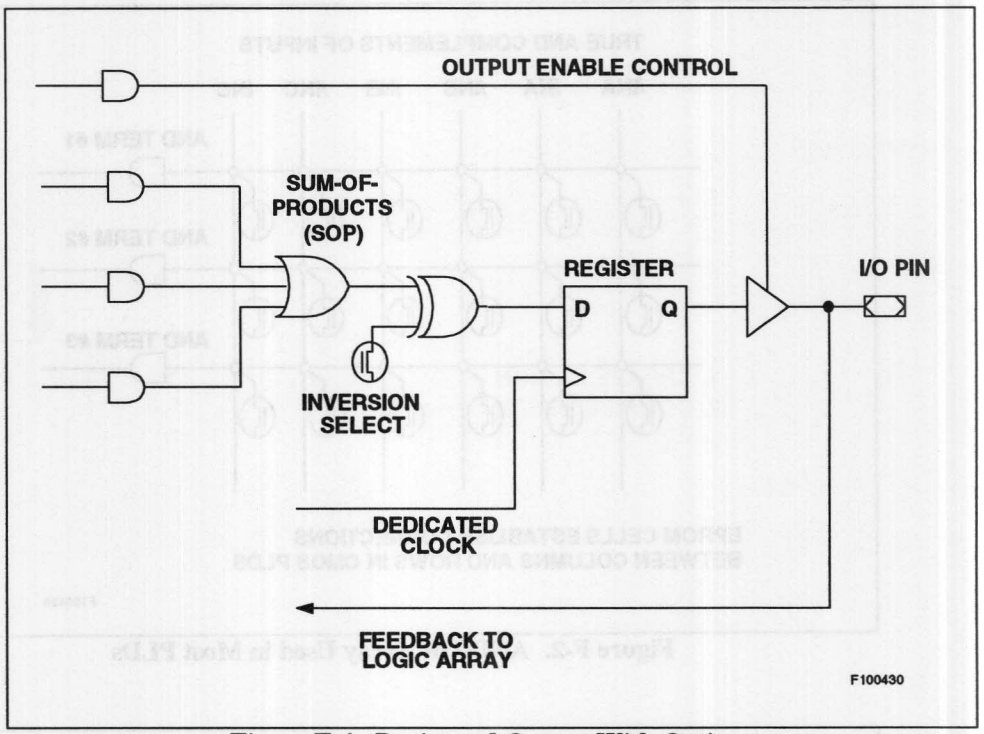


Figure F-4. Registered Output With Options

(billions of EPROM cells). These erasable cells allow the entire device to be programmed and tested during manufacturing to guarantee product quality. Bipolar technology, with its one-time-programmable fuses, cannot be fully tested during manufacturing.

CMOS PLDs also consume less power than bipolar PLDs and, therefore, generate less heat. This increases the reliability of your design and may allow you to use a less expensive power supply. CMOS PLDs have better metastability characteristics than bipolar PLDs, which can also increase system reliability.

PLD Design Process

A typical process for designing with PLDs, shown in Figure F-5, is as follows:

1. Logic to be implemented in a PLD is expressed in a source file using a design language. Design languages typically include Boolean equations, truth tables, a form of state machine syntax, and a functional simulation syntax.

2. The file is processed by a logic compiler to generate a JEDEC file representation of the design. The compiler configures the bits in the JEDEC file that make/break connections in the logic array and configure the programmable features on the outputs. Simulation can also be performed during processing.

3. This JEDEC file is then used to program the PLD for its target application.

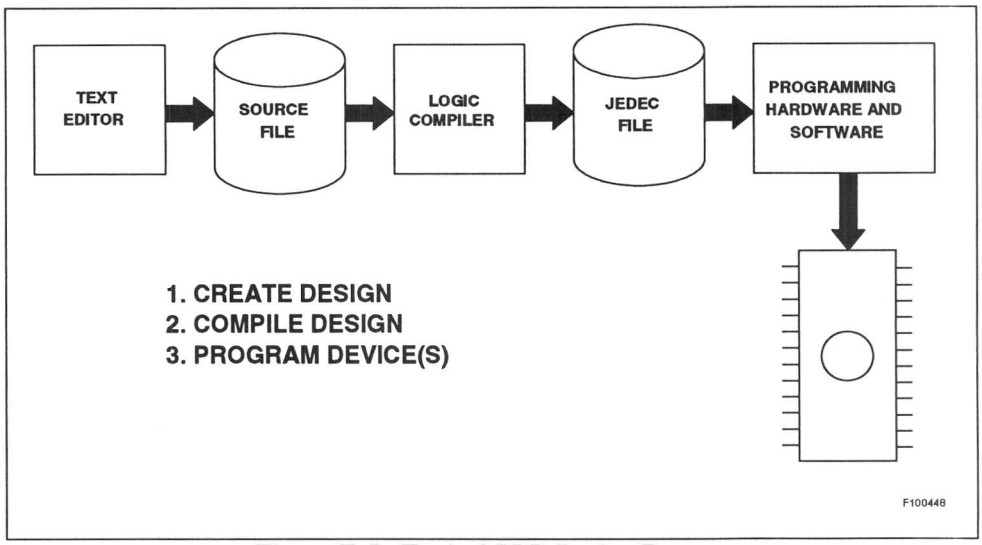


Figure F-5. Typical PLD Design Process

The task of creating the source file involves knowledge of (1) the target application requirements, (2) the device features, and (3) the way to implement device features in the source file.

For example, let's assume that you need a simple circuit that would be represented in schematic form as shown in the top half of Figure F-6. You choose a PLD with combinatorial outputs.

The logic for this design can be expressed in the following Boolean equations:

```
OUTA = INA * INB
OUTB = INA * /INB
```

The first equation means, "when INA and INB are both high, drive OUTA high." The second equation means, "when INA is high and INB is low, drive OUTB high." The compiler processes these equations, generating the JEDEC bit values that will connect AND terms to the proper true and complement input signals to implement these functions.

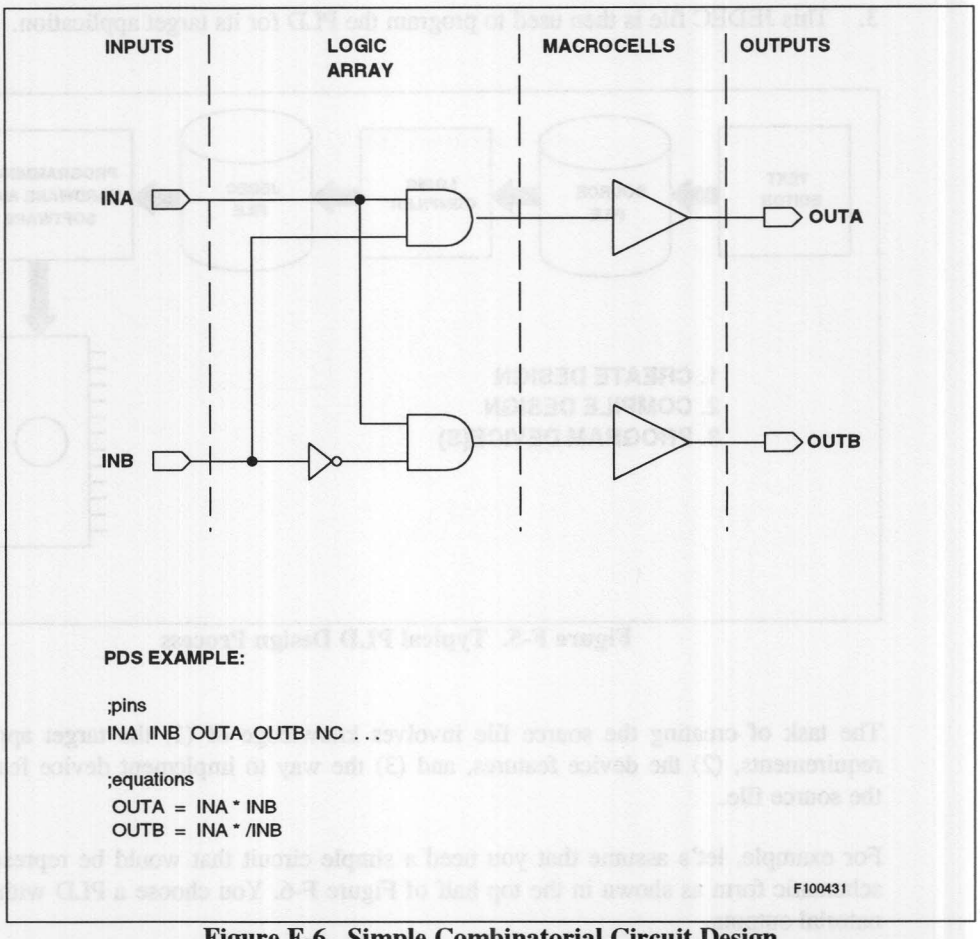


Figure F-6. Simple Combinatorial Circuit Design

When the output signal has the inversion prefix "/", the compiler automatically generates the proper JEDEC bits to invert the signal (inversion select bit). An active-low version of the second equation is as follows:

```
/OUTB = INA * /INB
```

The output can also be inverted by placing the inversion prefix "/" in the pin list for the desired signal.

Other programmable features in PLDs are automatically selected by the compiler based on the form the equations take. For example, when the equal sign (which selects a combinatorial output) is changed to a colon + equal sign (:=), as follows:

```
/OUTA := INA * /INB
```

the compiler understands this as a registered output and generates the appropriate JEDEC pattern to implement the registered option. Figure F-7 shows this register in schematic and equation form.

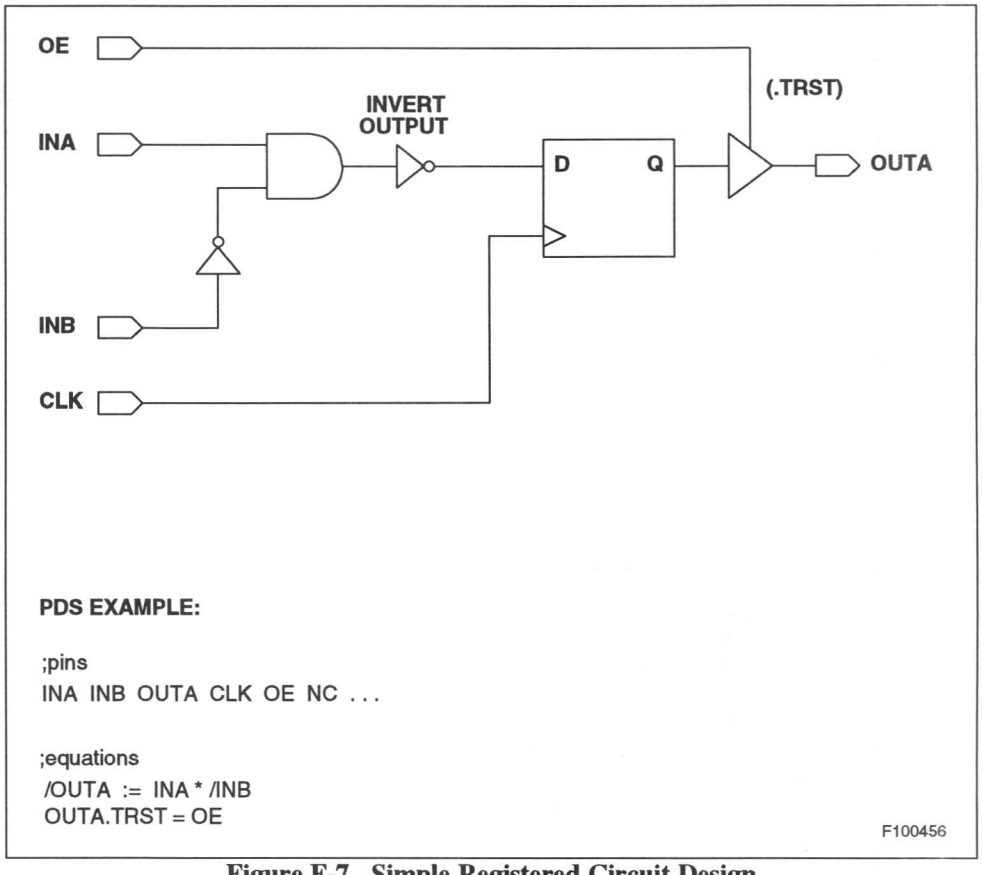


Figure F-7. Simple Registered Circuit Design

If an output enable (OE) p-term is needed to enable/disable the output buffer for this register, the same output name is used, but an extension is added, as follows:

```
/OUTA := INA * /INB
OUTA.TRST = OE
```

The ".TRST" extension indicates the "three-state" or high-impedance p-term. Other control signals, such as Clears, Presets, or Asynchronous Clocks are implemented in a similar manner (output name + appropriate extension).

If a simulation section is added at the end of a PLDasm source file, the PLDasm simulator will simulate the design and create a file to show the results of the simulation. Figure F-8 illustrates a simulation of a simple combinatorial design. This output file can then be checked to see if the circuit is functioning properly.

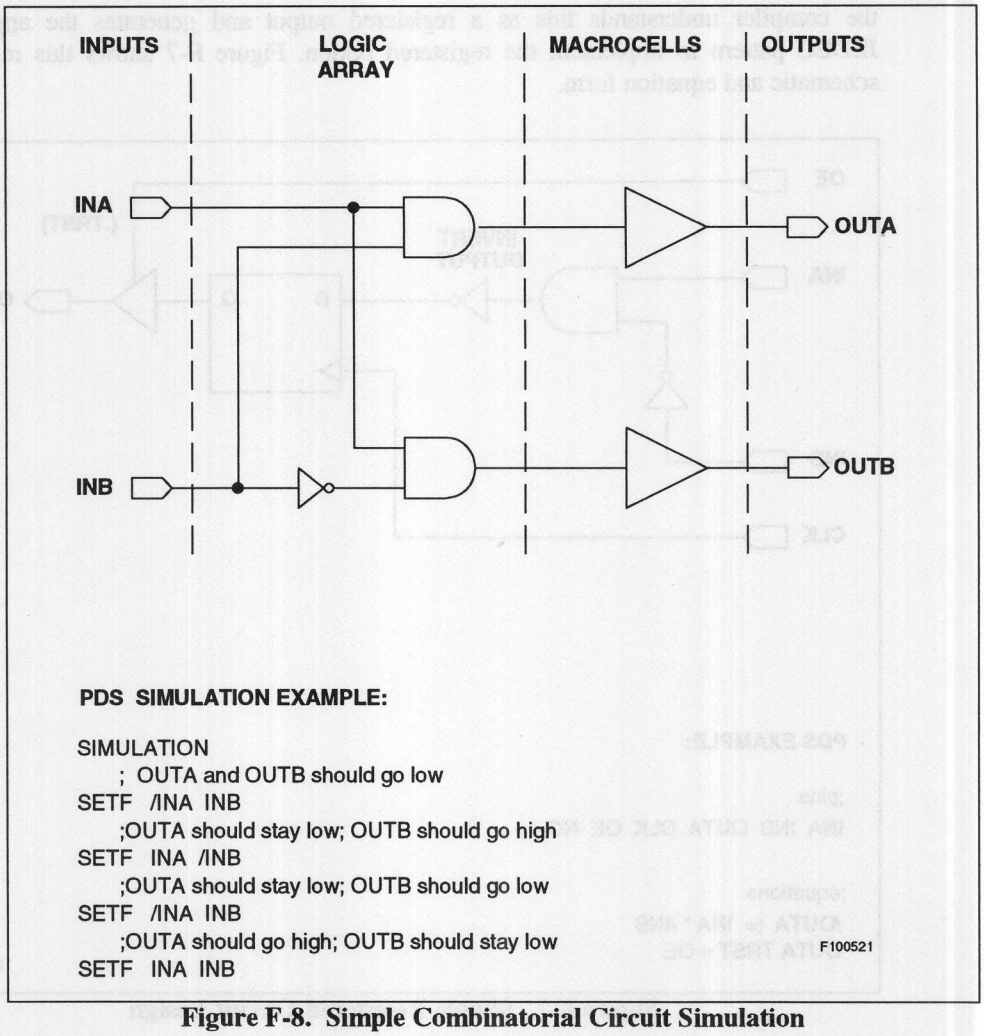


Figure F-8. Simple Combinatorial Circuit Simulation

The example uses Boolean equation syntax to implement the design. Two other types of syntax are supported for designs: State Machine syntax, and Truth Table syntax. These different ways of describing designs allow designers to choose the method most familiar to them and best suited for the target application.

The main body of this user's guide describes the features of Intel µPLDs and shows how to take advantage of those features in PLDasm syntax.

Index

S